

Tanja Lange · Rainer Steinwandt (Eds.)

LNCS 10786

Post-Quantum Cryptography

9th International Conference, PQCrypto 2018
Fort Lauderdale, FL, USA, April 9–11, 2018
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, Chennai, 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>

Tanja Lange · Rainer Steinwandt (Eds.)

Post-Quantum Cryptography

9th International Conference, PQCrypto 2018
Fort Lauderdale, FL, USA, April 9–11, 2018
Proceedings

Editors

Tanja Lange
Technische Universiteit Eindhoven
Eindhoven
The Netherlands

Rainer Steinwandt
Florida Atlantic University
Boca Raton, FL
USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-79062-6 ISBN 978-3-319-79063-3 (eBook)
<https://doi.org/10.1007/978-3-319-79063-3>

Library of Congress Control Number: 2018937396

LNCS Sublibrary: SL4 – Security and Cryptology

© Springer International Publishing AG, part of Springer Nature 2018

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 the registered company Springer International Publishing AG
part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

PQCrypto 2018, the 9th International Conference on Post-Quantum Cryptography, was held in Fort Lauderdale, Florida, USA, during April 9–11, 2018.

The aim of the PQCrypto conference series is to serve as a forum for researchers to present results and exchange ideas on cryptography in an era with large-scale quantum computers.

PQCrypto 2018 was co-located with NIST’s First PQC Standardization Conference (April 11–13, 2018) in Fort Lauderdale, Florida. Following the same model as its predecessor, PQCrypto 2018 adopted a two-stage submission process in which authors registered their paper one week before the final submission deadline.

The conference received 97 submissions with authors from 30 countries. Each paper (that had not been withdrawn by the authors) was reviewed in private by at least three Program Committee members. The private review phase was followed by an intensive discussion phase, conducted online. At the end of this process, the Program Committee selected 24 papers for inclusion in the technical program and publication in these proceedings. The accepted papers cover a broad spectrum of research within the conference’s scope, including code-, hash-, isogeny-, and lattice-based cryptography, multivariate cryptography, and quantum cryptanalysis.

Along with the 24 contributed technical presentations, the program featured outstanding invited talks and a presentation on NIST’s post-quantum cryptography standardization.

Organizing and running this year’s edition of the PQCrypto conference series was a team effort, and we are indebted to everyone who helped make PQCrypto 2018 a success. In particular, we would like to thank all members of the Program Committee and the external reviewers who were vital for compiling the technical program. Evaluating and discussing the submissions was a labor-intensive task, and we truly appreciate the work that went into this. We also owe a big thank you to Maria Provost from Florida Atlantic University, who made sure that all local arrangements fell into place as needed.

February 2018

Tanja Lange
Rainer Steinwandt

PQCrypto 2018

9th International Conference on Post-Quantum Cryptography

Fort Lauderdale, Florida, USA
April 9–11, 2018

Program Chairs

Tanja Lange
Rainer Steinwandt

Technische Universiteit Eindhoven, The Netherlands
Florida Atlantic University, USA

Steering Committee

Daniel J. Bernstein
Johannes Buchmann
Claude Crépeau
Jintai Ding
Philippe Gaborit
Tanja Lange
Daniele Micciancio
Michele Mosca
Nicolas Sendrier
Tsuyoshi Takagi
Bo-Yin Yang

University of Illinois at Chicago, USA
Technische Universität Darmstadt, Germany
McGill University, Canada
University of Cincinnati, USA
University of Limoges, France
Technische Universiteit Eindhoven, The Netherlands
University of California at San Diego, USA
University of Waterloo, Canada
Inria, France
Kyushu University and University of Tokyo, Japan
Academia Sinica, Taiwan

Program Committee

Gorjan Alagic
Shi Bai
Lejla Batina
Daniel J. Bernstein
Joppe Bos
Johannes Buchmann
Wouter Castryck
Pierre-Louis Cayrel
Chen-Mou Cheng
Jung Hee Cheon
Andrew Childs
Jintai Ding
Thomas Eisenbarth

University of Maryland, USA
Florida Atlantic University, USA
Radboud University, The Netherlands
University of Illinois at Chicago, USA
NXP Semiconductors, Belgium
Technische Universität Darmstadt, Germany
KU Leuven, Belgium
Université Jean Monnet de Saint-Etienne, France
Osaka University, Japan
Seoul National University, Korea
University of Maryland, USA
University of Cincinnati, USA
Universität zu Lübeck, Germany and Worcester
Polytechnic Institute, USA

Scott Fluhrer	Cisco Systems, USA
Philippe Gaborit	Université Limoges, France
Tommaso Gagliardoni	IBM Research, Switzerland
Kris Gaj	George Mason University, USA
Steven Galbraith	Auckland University, New Zealand
Tim Güneysu	Ruhr-Universität Bochum and DFKI, Germany
Sean Hallgren	Pennsylvania State University, USA
Yasufumi Hashimoto	University of the Ryukyus, Japan
Andreas Hülsing	Technische Universiteit Eindhoven, The Netherlands
David Jao	University of Waterloo and evolutionQ, Inc., Canada
Stacey Jeffery	CWI and QuSoft, The Netherlands
Thomas Johansson	Lund University, Sweden
Kwangjo Kim	KAIST, Korea
Stefan Kölbl	Technical University of Denmark, Denmark
Tancrède Lepoint	SRI International, USA
Yi-Kai Liu	NIST, USA
Michele Mosca	University of Waterloo and Perimeter Institute, Canada
Michael Naehrig	Microsoft Research, USA
María Naya-Plasencia	Inria, France
Ruben Niederhagen	Fraunhofer SIT, Germany
Edoardo Persichetti	Florida Atlantic University, USA
Thomas Pöppelmann	Infineon Technologies, Germany
Christian Rechberger	University of Graz, Austria
Martin Roetteler	Microsoft Research, USA
Alexander Russell	University of Connecticut, USA
Simona Samardjiska	Radboud Universiteit, The Netherlands and Ss. Cyril and Methodius University, Macedonia
Peter Schwabe	Radboud Universiteit, The Netherlands
Nicolas Sendrier	Inria, France
Daniel Smith-Tone	NIST and University of Louisville, USA
Fang Song	Portland State University, USA
Douglas Stebila	McMaster University, Canada
Damien Stehlé	ENS de Lyon, France
Krysta Svore	Microsoft Research, USA
Tsuyoshi Takagi	Kyushu University and University of Tokyo, Japan
Jean-Pierre Tillich	Inria, France
Christine van Vredendaal	Technische Universiteit Eindhoven, The Netherlands
William Whyte	OnBoard Security, USA
Keita Xagawa	NTT, Japan
Bo-Yin Yang	Academia Sinica, Taiwan
Zhang Zhenfeng	Chinese Academy of Sciences, China

External Reviewers

Martin R. Albrecht	Seungwan Hong	Chris Peikert
Jacob Alperin-Sheriff	Yasuhiko Ikematsu	Geovandro Pereira
Koichiro Akiyama	Ilia Iliashenko	Ray Perlner
Reza Azarderakhsh	Brian Jarvis	Peter Pessl
Gustavo Banegas	Haodong Jiang	Christophe Petit
Ward Beullens	Jinhyuck Jeong	Albrecht Petzoldt
Jean-Francois Biasse	Stephen Jordan	Alexander Poremba
Nina Bindel	Elif Bilge Kavun	Yuming Qiao
Cecilia Boschini	Andrey Kim	Somindu C. Ramanna
Charles Bouillaguet	Duhyeong Kim	Joost Renes
Kevin Carrier	Jae-yun Kim	Adeline Roux-Langlois
Ryann Cartor	Jiseung Kim	Paolo Santini
Cong Chen	Elena Kirshanova	Pascal Sasdrich
Long Chen	Jean Belo Klamti	John Schanck
Rakyong Choi	Norman Lahr	Kevin Schmidt
Tung Chou	Joohee Lee	Gregor Seiler
Craig Costello	Jason LeGrow	Yongha Son
Joan Daemen	Matthieu Lequesne	Yongsoo Song
Viet Ba Dang	Aaron Lye	Vladimir Soukharev
Thomas Debris-Alazard	Christian Majenz	Paul Stankovski
Luca De Feo	Chloe Martindale	Alan Szepieniec
Javad Doliskani	Pedro Maat Massolino	Mostafa Taha
Léo Ducas	Dustin Moody	Atsushi Takayasu
Dung Hoang Duong	Travis Morrison	Guofeng Tang
Kirsten Eisentraeger	Koksal Mus	Yan Bo Ti
Ahmed Ferozपुरi	Duc Tri Nguyen	Patrick Towa
Xinwei Gao	Khoa Nguyen	Nguenewou
Nicholas Genise	Phong Nguyen	Jeremy Vates
Vlad Gheorghiu	Xuyun Nie	Valentin Vasseur
Lorenzo Grassi	Tobias Oder	Yacheng Wang
Kyoohyung Han	Ayoub Otmani	Weiqiang Wen
Christian Hanser	Yanbin Pan	Shih-Chun You
Minki Hhan	Lorenz Panny	Mark Zhandry

Contents

Code-Based Cryptography

LEDAkem: A Post-quantum Key Encapsulation Mechanism Based on QC-LDPC Codes	3
<i>Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini</i>	
Decoding Linear Codes with High Error Rate and Its Impact for LPN Security.	25
<i>Leif Both and Alexander May</i>	
QC-MDPC: A Timing Attack and a CCA2 KEM	47
<i>Edward Eaton, Matthieu Lequesne, Alex Parent, and Nicolas Sendrier</i>	
FPGA-Based Niederreiter Cryptosystem Using Binary Goppa Codes	77
<i>Wen Wang, Jakub Szefer, and Ruben Niederhagen</i>	

Cryptanalysis

Attacks on the AJPS Mersenne-Based Cryptosystem	101
<i>Koen de Boer, Léo Ducas, Stacey Jeffery, and Ronald de Wolf</i>	
Implementing Joux-Vitse’s Crossbred Algorithm for Solving MQ Systems over \mathbb{F}_2 on GPUs	121
<i>Ruben Niederhagen, Kai-Chun Ning, and Bo-Yin Yang</i>	
Practical Cryptanalysis of a Public-Key Encryption Scheme Based on Non-linear Indeterminate Equations at SAC 2017.	142
<i>Keita Xagawa</i>	

Hash-Based Cryptography

Grafting Trees: A Fault Attack Against the SPHINCS Framework.	165
<i>Laurent Castelnovi, Ange Martinelli, and Thomas Prest</i>	
Post-quantum Security of the Sponge Construction	185
<i>Jan Czajkowski, Leon Groot Bruinderink, Andreas Hülsing, Christian Schaffner, and Dominique Unruh</i>	
Putting Wings on SPHINCS	205
<i>Stefan Kölbl</i>	

Isogenies in Cryptography

Computing Isogenies Between Montgomery Curves
Using the Action of $(0, 0)$ 229
Joost Renes

Faster Isogeny-Based Compressed Key Agreement 248
*Gustavo H. M. Zanon, Marcos A. Simplicio Jr,
Geovandro C. C. F. Pereira, Javad Doliskani,
and Paulo S. L. M. Barreto*

Lattice-Based Cryptography

Practical Implementation of Ring-SIS/LWE Based Signature and IBE 271
*Pauline Bert, Pierre-Alain Fouque, Adeline Roux-Langlois,
and Mohamed Sabt*

Progressive Lattice Sieving. 292
Thijs Laarhoven and Artur Mariano

A Nonstandard Variant of Learning with Rounding with Polynomial
Modulus and Unbounded Samples 312
Hart Montgomery

Lattice-Based Signcryption Without Random Oracles 331
Shingo Sato and Junji Shikata

Multivariate Cryptography

Rank Analysis of Cubic Multivariate Cryptosystems 355
*John Baena, Daniel Cabarcas, Daniel E. Escudero, Karan Khathuria,
and Javier Verbel*

Improved Cryptanalysis of HFEv- via Projection. 375
Jintai Ding, Ray Perlner, Albrecht Petzoldt, and Daniel Smith-Tone

HFERP - A New Multivariate Encryption Scheme 396
*Yasuhiko Ikematsu, Ray Perlner, Daniel Smith-Tone, Tsuyoshi Takagi,
and Jeremy Vates*

Protocols

Post-Quantum Zero-Knowledge Proofs for Accumulators with Applications
to Ring Signatures from Symmetric-Key Primitives. 419
David Derler, Sebastian Ramacher, and Daniel Slamanig

G-Merkle: A Hash-Based Group Signature Scheme
 from Standard Assumptions 441
Rachid El Bansarkhani and Rafael Misoczki

Quantum Algorithms

Quantum Collision-Finding in Non-uniform Random Functions 467
Marko Balogh, Edward Eaton, and Fang Song

Asymptotically Faster Quantum Algorithms to Solve Multivariate
 Quadratic Equations 487
Daniel J. Bernstein and Bo-Yin Yang






Improved Quantum Information Set Decoding 507
Elena Kirshanova

Author Index 529

Code-Based Cryptography



LEDAkem: A Post-quantum Key Encapsulation Mechanism Based on QC-LDPC Codes

Marco Baldi¹ , Alessandro Barenghi² , Franco Chiaraluce¹ ,
Gerardo Pelosi² , and Paolo Santini¹ 

¹ Università Politecnica delle Marche, Ancona, Italy
{m.baldi,f.chiaraluce}@univpm.it, p.santini@pm.univpm.it

² Politecnico di Milano, Milan, Italy
{alessandro.barenghi,gerardo.pelosi}@polimi.it

Abstract. This work presents a new code-based key encapsulation mechanism (KEM) called LEDAkem. It is built on the Niederreiter cryptosystem and relies on quasi-cyclic low-density parity-check codes as secret codes, providing high decoding speeds and compact keypairs. LEDAkem uses ephemeral keys to foil known statistical attacks, and takes advantage of a new decoding algorithm that provides faster decoding than the classical bit-flipping decoder commonly adopted in this kind of systems. The main attacks against LEDAkem are investigated, taking into account quantum speedups. Some instances of LEDAkem are designed to achieve different security levels against classical and quantum computers. Some performance figures obtained through an efficient C99 implementation of LEDAkem are provided.

Keywords: Code-based cryptography
Key encapsulation mechanism · Niederreiter cryptosystem
Post-quantum cryptography
Quasi-cyclic low-density parity-check codes

1 Introduction

Devising efficient and robust post-quantum key encapsulation mechanisms (KEMs) is an important and urgent research target, as also witnessed by the recent NIST call for post-quantum cryptographic systems [32]. Code-based cryptosystems are among the most promising candidates to replace quantum-vulnerable primitives which are still relying on the hardness of the integer factorization or discrete logarithm problems, such as the Diffie-Hellman key exchange and the Rivest-Shamir-Adleman (RSA) and ElGamal cryptosystems. Indeed, Shor's algorithm [41] can be used to solve both the integer factorization and the discrete logarithm problems in polynomial time with a quantum computer. One of the problems for which no known polynomial time algorithm on a quantum

computer exists is the decoding of a general linear code. Indeed, such a problem belongs to the non deterministic-polynomial (NP)-complete computational equivalence class [11, 27], which is widely believed to contain problems which have no polynomial time solution on a quantum computer.

The first code-based public-key cryptosystem relying on the general linear code decoding problem was proposed by McEliece in 1978 [28], and used Goppa codes [18] to form the secret key. Such a choice yields large public keys, which is the main limitation of Goppa code-based systems. The Niederreiter cryptosystem [34] is a code-based cryptosystem exploiting the same trapdoor, but using syndromes and parity-check matrices instead of codewords and generator matrices as in McEliece. When the same family of codes is used, Niederreiter and McEliece are equivalent [25] and therefore they achieve the same security levels.

Replacing Goppa codes with other families of more structured codes may reduce the public key size. However, this may also compromise the system security, as it occurred with some first McEliece variants based on quasi-cyclic (QC) codes [17], low-density parity-check (LDPC) codes [31] and quasi-cyclic low-density parity-check (QC-LDPC) codes [35], quasi-dyadic (QD) codes [30], convolutional codes [26] and some instances based on generalized Reed-Solomon (GRS) codes [7, 10]. Nevertheless, some variants exploiting QC-LDPC and quasi-cyclic moderate-density parity-check (QC-MDPC) codes [2, 3, 29] have been shown to be able to achieve very compact keys without endangering security.

Recently, some new statistical attacks have been developed that exploit the information coming from decryption failures in QC-LDPC and QC-MDPC code-based systems to perform key recovery attacks, thus forcing to renew keys frequently in these systems [16, 20].

In this paper, we start from the QC-LDPC code-based system proposed in [2, 3] and we develop a new KEM based on the the Niederreiter cryptosystem. We also introduce an improved decoding algorithm which exploits correlation among intentional errors seen by the private code. This way, the correction capability of the private code is exploited to the utmost, thus allowing to achieve significant reductions in the public key size. We call the new system LEDAkem and study its properties and security. We take into account the fact that Grover's algorithm running on a quantum computer may be exploited to speedup attacks based on information set decoding (ISD) [22, 43], and we propose some sets of parameters for LEDAkem achieving different security levels against attacks exploiting both classical and quantum computers. We also describe an optimized software implementation of the proposed system and provide and discuss some performance figures. LEDAkem currently is one of the first round candidate algorithms of the NIST post-quantum cryptography standardization project [32], along with other code-based KEMs. In this work we will highlight the differences between our proposal and the closest one among the others, i.e. BIKE [1], which relies on QC-MDPC codes for its construction.

The organization of the paper is as follows. In Sect. 2 we describe LEDAkem. In Sect. 3 we present its security analysis and in Sect. 4 its peculiar features. In Sect. 5 we discuss some implementation issues and we show some numerical results. Finally, some conclusions are drawn in Sect. 6.

2 The LEDAkem Cryptosystem

The LEDAkem cryptosystem is derived from the Niederreiter cryptosystem with the following main differences:

- Non-algebraic codes known as QC-LDPC codes are used as secret codes.
- The public code is neither coincident with nor equivalent to the private code.
- Suitably designed iterative non-bounded-distance decoding algorithms are used.

The motivation for using QC-LDPC codes as private codes is in the fact that these codes are known to achieve important reductions in the public key size when used in this context [2, 29]. Moreover, when LDPC codes are used as private codes, the public code cannot be either coincident with or equivalent to the private code. Indeed, in such a case, an attacker could search for low weight codewords in the dual of the public code and find a sparse parity-check matrix of the private code which allows efficient decoding.

For this reason, following [2], LEDAkem uses a transformation matrix Q that hides the sparse parity-check matrix H of the private code into a denser parity-check matrix $L = HQ$ of the public code. This also affects the error vector that must be corrected during decryption, which is obtained from the error vector used during encryption through multiplication by Q . In this work, we show how it is possible to exploit the knowledge of Q to design an ad-hoc decoding algorithm achieving very good performance in terms of both decoding speed and decryption failure rate (DFR).

In fact, a well-known feature of LDPC coding is that the decoding radius of iterative decoders is not sharp and cannot be estimated in a deterministic way. It follows that some residual DFR must be tolerated, and it must be estimated heuristically through Montecarlo simulations. This is done for all the proposed instances of LEDAkem in order to guarantee that they achieve a sufficiently low DFR. Providing quantitative estimates of the DFR for the proposed instances of LEDAkem allows us to prevent attacks such as the ones described in [16, 20] changing the key either at each round of the KEM, or before a sufficient amount of decoding failures are observed by the attacker.

2.1 Coding Background

A QC code is defined as a linear block code with dimension $k = pk_0$ and length $n = pn_0$, in which each cyclic shift of a codeword by n_0 symbols results in another valid codeword. It follows from their definition that QC codes have generator and parity-check matrices in “blocks circulant” form or, equivalently, in “circulants block” form. The latter is used in LEDAkem. A $v \times v$ circulant matrix A has the following form

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{v-1} \\ a_{v-1} & a_0 & a_1 & \cdots & a_{v-2} \\ a_{v-2} & a_{v-1} & a_0 & \cdots & a_{v-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_0 \end{bmatrix}. \quad (1)$$

According to its definition, any circulant matrix is regular, since all its rows and columns are cyclic shifts of the first row and column, respectively.

The set of $v \times v$ binary circulant matrices forms an algebraic ring under the standard operations of modulo-2 matrix addition and multiplication. The zero element is the all-zero matrix, and the identity element is the $v \times v$ identity matrix. The algebra of the polynomial ring $\mathbb{F}_2[x]/\langle x^v + 1 \rangle$ is isomorphic to the ring of $v \times v$ circulant matrices over \mathbb{F}_2 with the following map

$$A \leftrightarrow a(x) = \sum_{i=0}^{v-1} a_i x^i. \quad (2)$$

According to (2), any binary circulant matrix is associated to a polynomial in the variable x having coefficients over \mathbb{F}_2 which coincide with the entries of the first row of the matrix

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{v-1} x^{v-1}. \quad (3)$$

According to (2), the all-zero circulant matrix corresponds to the null polynomial and the identity matrix to the unitary polynomial. The ring of polynomials $\mathbb{F}_2[x]/\langle x^v + 1 \rangle$ includes elements that are zero divisors which are mapped to singular circulant matrices over \mathbb{F}_2 . Avoiding such matrices is important in some parts of LEDAkem, and smart ways exist to design non-singular circulant matrices. As it will be described next, the main part of the secret key of LEDAkem is formed by a binary QC-LDPC code described through its parity-check matrix H . Let n denote the code length in bits and k denote the code dimension in bits, then H has size $(n - k) \times n = r \times n$, where r is the code redundancy.

2.2 Description of the Primitives

The main functions of LEDAkem are described next.

Key Generation. Both private and public keys consist of binary matrices. These matrices, in their turn, are formed by $p \times p$ circulant blocks, being p an integer properly chosen.

Secret key. The key generation input is formed by:

- The circulant block size p (usually in the order of some thousands bits).
- The integer n_0 (usually between 2 and 4), representing the number of circulant blocks forming the matrix H .

- The integer d_v , representing the row/column weight (usually between 15 and 25) of the circulant blocks forming the matrix H .
- The vector of integers $\bar{m} = [m_0, m_1, \dots, m_{n_0-1}]$, representing the row/column weights (each entry usually smaller than 10) of the circulant blocks forming the matrix Q (the structure of Q is clarified below).

Given these inputs, the secret key is obtained as follows.

First, n_0 sparse circulant matrices with size $p \times p$ are generated at random. Each of them has row/column weight d_v . We denote such matrices as $H_0, H_1, \dots, H_{n_0-1}$. The secret low-density parity-check matrix H is then obtained as

$$H = [H_0 | H_1 | H_2 | \dots | H_{n_0-1}]. \quad (4)$$

The size of H is $p \times n_0 p$. Other n_0^2 sparse circulant blocks $Q_{i,j}$ are then randomly generated to form the secret sparse matrix

$$Q = \begin{bmatrix} Q_{0,0} & Q_{0,1} & \dots & Q_{0,n_0-1} \\ Q_{1,0} & Q_{1,1} & \dots & Q_{1,n_0-1} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{n_0-1,0} & Q_{n_0-1,1} & \dots & Q_{n_0-1,n_0-1} \end{bmatrix}. \quad (5)$$

The row/column weight of each block $Q_{i,j}$ is fixed according to the following matrix

$$w(Q) = \begin{bmatrix} m_0 & m_1 & \dots & m_{n_0-1} \\ m_{n_0-1} & m_0 & \dots & m_{n_0-2} \\ \vdots & \vdots & \ddots & \vdots \\ m_1 & m_2 & \dots & m_0 \end{bmatrix}, \quad (6)$$

such that each row and each column of Q has weight $m = \sum_{i=0}^{n_0-1} m_i$.

The choice of the weights $\bar{m} = [m_0, m_1, \dots, m_{n_0-1}]$ and the size p of the circulant blocks composing it is very important since it allows to discern if Q is invertible or not. In particular, denoting with $\mathbf{\Pi}\{\cdot\}$ the permanent of a matrix, the following theorem holds.

Theorem 1. *Let $p > 2$ be a prime such that $\text{ord}_p(2) = p-1$ and Q be an $n_0 \times n_0$ matrix with elements in $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$; if $\mathbf{\Pi}\{w(Q)\}$ is odd and $\mathbf{\Pi}\{w(Q)\} < p$, then Q is non singular.*

Proof. Omitted for the sake of brevity.

With this result, we can guarantee that, when the sequence \bar{m} is properly chosen, the matrix Q is always non singular, which is a necessary condition for the key generation process to be successful.

Definition 1. *The secret key (SK) of LEDAkem is formed by $\{H, Q\}$.*

Since both H and Q are formed by sparse circulant blocks, it is convenient to represent each of these blocks through the indexes of the symbols 1 in their first row, i.e., adopt a sparse representation for them. Each index of this type requires $\lceil \log_2(p) \rceil$ bits to be stored. If we consider that the circulant blocks in any block row of Q have overall weight $m = \sum_{i=0}^{n_0-1} m_i$, the size of SK in bits is

$$S_{sk} = n_0 (d_v + m) \lceil \log_2(p) \rceil. \quad (7)$$

In practice, the secret matrices are generated through a deterministic random bit generator (DRBG), seeded with a bit string extracted from a true random number generator (TRNG). In this case, to obtain H and Q it is sufficient to know the TRNG extracted seed of the DRBG that has been used to generate the positions of their non-null coefficients, since this process is rather fast. This approach allows reducing the size of the secret key to the minimum required, as it is assumed that the TRNG output cannot be compressed. The entity of the reduction depends on the values of the parameters involved in (7).

Public key. Starting from H and Q , the following binary matrices are computed. First of all, the matrix L is obtained as

$$L = HQ = [L_0|L_1|L_2|\dots|L_{n_0-1}]. \quad (8)$$

If both d_v and m are odd, then L_{n_0-1} has full-rank. In fact, $L_{n_0-1} = \sum_{i=0}^{n_0-1} H_i Q_{i,n_0-1}$ and has weight equal to $md_v - 2c$ (where c is the number of cancellations occurred in the product). It is possible to demonstrate that if md_v is odd and $md_v < p$ then L_{n_0-1} is non-singular.

After inverting L_{n_0-1} , the following matrix is computed:

$$M = L_{n_0-1}^{-1}L = [M_0|M_1|M_2|\dots|M_{n_0-2}|I] = [M_l|I]. \quad (9)$$

Definition 2. *The public key (PK) of LEDAkem is formed by $M_l = [M_0|M_1|M_2|\dots|M_{n_0-2}]$.*

Since the circulant blocks forming M_l are dense, it is convenient to store them through the binary representation of their first row (the other rows are then obtained as cyclic shifts of the first row). The bit-size of the PK hence is

$$S_{pk} = (n_0 - 1)p. \quad (10)$$

Encryption. The plaintext of LEDAkem is an ephemeral random secret generated by Bob who is willing to share it with Alice. The encryption inputs are:

- The values of n_0 and p , from which $n = n_0p$ is computed.
- The number of intentional errors $t \ll n$.

Bob generates a secret in the form of a random binary vector e with length of $n = n_0p$ bits and Hamming weight t . Given a key derivation function (KDF), the shared secret key k_s is generated from e as $k_s = \text{KDF}(e)$. In order to encapsulate the shared secret e , Bob fetches Alice's PK M_l and computes $s = [M_l|I] e^T$ where T denotes matrix transposition. The $p \times 1$ syndrome vector s representing the encapsulated secret is then sent to Alice.

Decryption. In order to perform decryption, Alice must recover e from s . The latter can be written as $s = Me^T = L_{n_0-1}^{-1}Le^T = L_{n_0-1}^{-1}HQe^T$. The first decryption step for Alice is computing $s' = L_{n_0-1}s = HQe^T$. For this purpose, Alice needs to know L_{n_0-1} that, according to (8), is the last circulant block of the matrix HQ . Hence, it can be easily computed from the SK which contains both H and Q . If we define the *expanded error vector* as

$$e' = eQ^T, \quad (11)$$

then we have $s' = He'^T$. Hence, QC-LDPC decoding through H can be exploited for recovering e' from s' . QC-LDPC decoders are not bounded distance decoders, and some DFR must be tolerated. However, the system parameters can be chosen such that the DFR is acceptably small. For this purpose, the average decoding radius of the private code must be sufficiently larger than the Hamming weight of e' , which is approximately equal to mt (due to the sparsity of Q and e). Then, multiplication by $(Q^T)^{-1}$ would be needed to obtain e from e' , that is,

$$e = e'(Q^T)^{-1}. \quad (12)$$

However, by exploiting the efficient decoding algorithm described in Sect. 2.3, this last step can be avoided, which also allows avoiding the computation and storage of $(Q^T)^{-1}$ as part of the secret key. In fact, the decoding algorithm described in Sect. 2.3 allows recovering e directly by performing decoding of $s' = L_{n_0-1}s = HQe^T$ through H , while taking into account the effect of the multiplication of e by Q . Then, the secret key is recovered as $k_s = \text{KDF}(e)$.

In case a decoding error occurs, the decryption procedure derives the shared secret combining with a KDF the syndrome with a secret constant, which may be derived via a PRNG from the secret key material [38]. Alternatively, using a secret permutation of the syndrome as input to the KDF was noted to be effective in [21]. Such an approach is beneficial from the security standpoint in case of an accidental keypair reuse. More details concerning this aspect, which is related to formal security of LEDAkem, will be given in Sect. 4. According to this approach, Bob will become aware of the decoding failure upon reception of the message sent by Alice encrypted with the incorrectly derived shared secret.

2.3 Efficient Decoding

Classical bit flipping (BF) decoding works as follows. At each iteration, for each codeword bit position, the number of unsatisfied parity-check equations is computed, and if this number equals or exceeds a given threshold, then that bit is flipped. The decision threshold can be chosen in many ways, affecting the decoder performance, and it can be fixed or it can vary during iterations. A choice that often turns out to be optimal is to fix the threshold, at each iteration, as the maximum number of unsatisfied parity-check equations in which any codeword bit is involved. In fact, a codeword bit participating in a higher number of unsatisfied parity-check equations can be considered less reliable than a codeword bit

participating in a smaller number of unsatisfied parity-check equations. So, if the threshold is chosen in this way, the bits that are flipped are those that are most likely affected by errors.

Starting from classical BF, we have developed an improved decoder that is specifically designed for LEDAkem, where the position of the ones in the expanded error vector e' to be corrected is influenced by the value of Q^T , as e' is equivalent to a random error vector e with weight t multiplied by Q^T . Since this improved decoder takes into account such a multiplication by the transpose of matrix Q to estimate with greater efficiency the locations of the bits of the expanded error vector, we denote it as *Q-decoder*.

Inputs of the decoder are the syndrome s' and the matrices H and Q according to (4) and (5), respectively. The output of the decoder is a $1 \times n$ vector \hat{e} or a decoding failure, where \hat{e} represents the decoder estimate of the error vector e appearing in the equality $s' = HQe^T$. The decoding process performs a maximum of l_{max} iterations, where the l -th iteration processes $s^{(l-1)}$ and $\hat{e}^{(l-1)}$ (that is the values at the previous iteration) and outputs $s^{(l)}$ and $\hat{e}^{(l)}$. A threshold criterion is adopted to compute the positions in $\hat{e}^{(l)}$ that must be changed. The threshold values $b^{(l)}$ can be chosen in different ways and affect the decoder performance. In the next section we describe a simple and effective procedure to design such values. The decoder initialization is performed by setting $s^{(0)} = s'^T$ and $\hat{e}^{(0)} = 0_n$, where 0_n is the length- n vector with all-zero entries. It is important to note that $s^{(0)}$ (and, by extension, $s^{(l)}$) is a row vector. Moreover, let us consider that all multiplications are binary, except those denoted with '*', which are performed in the integer domain \mathbb{Z} . The l -th iteration of the Q-decoder performs the following operations:

- i. Compute $\Sigma^{(l)} = [\sigma_1^{(l)}, \sigma_2^{(l)}, \dots, \sigma_n^{(l)}] = s^{(l-1)} * H$, resulting in a vector of integers having entries between 0 and d_v .
- ii. Compute $R^{(l)} = [\rho_1^{(l)}, \rho_2^{(l)}, \dots, \rho_n^{(l)}] = \Sigma^{(l)} * Q$.
- iii. Define $\mathfrak{S}^{(l)} = \{v \in [1, n] \mid \rho_v^{(l)} \geq b^{(l)}\}$.
- iv. Update $\hat{e}^{(l-1)}$ as

$$\hat{e}^{(l)} = \hat{e}^{(l-1)} + \mathbf{1}_{\mathfrak{S}^{(l)}}$$

where $\mathbf{1}_{\mathfrak{S}^{(l)}}$ is a length- n binary vector with all-zero entries, except those indexed by $\mathfrak{S}^{(l)}$.

- v. Update the syndrome as

$$s^{(l)} = s^{(l-1)} + \sum_{v \in \mathfrak{S}^{(l)}} q_v H^T$$

where q_v is the v -th row of Q^T .

- vi. If the weight of $s^{(l)}$ is zero then stop decoding and return $\hat{e}^{(l)}$.
- vii. If $l < l_{max}$ then increment l and go back to step (i), otherwise stop decoding and return a decoding failure.

As in classical BF, the first step of this algorithm computes the vector $\Sigma^{(l)}$. Each entry of this vector counts the number of unsatisfied parity-check equations

corresponding to that bit position, and takes values in $\{0, \dots, d_v\}$. This evaluates the likelihood that the binary element of e' at the same position is equal to one. Differently from classical BF, in step (ii) the correlation $R^{(l)}$ between these likelihoods and the rows of Q^T is computed. In fact, the expanded error vector $e' = eQ^T$ can be written as the sum of the rows of Q^T indexed by the support of e , that is $e' = \sum_{j \in \Psi\{e\}} q_j$ where $\Psi\{e\}$ denotes the support of e .

Since both Q and e are sparse (that is, $m, t \ll n$), cancellations between ones in the sum are very unlikely. When the correlation between $\Sigma^{(l)}$ and a generic row q_v of Q^T is computed, two cases may occur:

- If $v \notin \Psi\{e\}$, then it is very likely that q_v has a very small number of common ones with all the rows of Q^T forming e' , hence the correlation is small.
- If $v \in \Psi\{e\}$, then q_v is one of the rows of Q^T forming e' , hence the correlation is large.

The main difference with classical BF is that, while in the latter all error positions are considered as independent, the Q-decoder exploits the correlation among expanded errors which is present in LEDAkem, since their positions are influenced by Q^T . This allows achieving important reductions in the number of decoding iterations. As a further advantage, this decoder allows recovering e , besides e' , without the need of computing and storing the inverse of the matrix Q^T . For this purpose, it is sufficient that, at each iteration, the Q-decoder flips the bits of the estimated error vector e that correspond to the correlations values overcoming the threshold.

2.4 Choice of the Q-decoder Decision Thresholds

One important aspect affecting performance of the Q-decoder is the choice of the threshold values against which the correlation is compared at each iteration. A natural choice is to set the threshold used at iteration l equal to the maximum value of the correlation $R^{(l)}$, that is $b^{(l)} = \max_{j=1,2,\dots,n} \{\rho_j^{(l)}\}$. This strategy ensures that only those few bits that have maximum likelihood of being affected by errors are flipped during each iteration, thus achieving the lowest DFR. However, such an approach has some drawbacks in terms of complexity, since the computation of the maximum correlation requires additional computations with respect to a fixed threshold.

Therefore, as in [14], we consider a different strategy, which allows computing the threshold values on the basis of the syndrome weight at each iteration. According to this approach, during an iteration it is sufficient to compute the syndrome weight and read the corresponding threshold value from a look-up table. This strategy still allows to achieve a sufficiently low DFR, while employing a significantly smaller number of decoding iterations.

Let us consider the l -th iteration of the Q-decoder, and denote by t_l the weight of the error vector $e^{(l)}$ and with t'_l the weight of the corresponding expanded error vector $e'^{(l)} = e^{(l)}Q^T$. Let us introduce the following probabilities [6]

$$\begin{aligned}
p_{ci}(t'_l) &= \sum_{j=0, j \text{ odd}}^{\min[n_0 d_v - 1, t'_l]} \frac{\binom{n_0 d_v - 1}{j} \binom{n - n_0 d_v}{t'_l - j}}{\binom{n-1}{t'_l}} \\
p_{ic}(t'_l) &= \sum_{j=0, j \text{ even}}^{\min[n_0 d_v - 1, t'_l - 1]} \frac{\binom{n_0 d_v - 1}{j} \binom{n - n_0 d_v}{t'_l - j - 1}}{\binom{n-1}{t'_l - 1}}
\end{aligned} \tag{13}$$

where $p_{ci}(t'_l)$ is the probability that a codeword bit is error-free and a parity-check equation evaluates it to be incorrect, and $p_{ic}(t'_l)$ is the probability that a codeword bit is error-affected and a parity-check equation evaluates it to be correct. In both these cases, the syndrome bit is equal to 1. The probability that each syndrome bit is equal to 1 can be therefore computed as $p_{ic}(t'_l) + p_{ci}(t'_l)$, so the average syndrome weight at iteration l results in

$$w_s^{(l)} = E \left[wt \left\{ s^{(l)} \right\} \right] = [p_{ic}(t'_l) + p_{ci}(t'_l)] p \tag{14}$$

where $wt \{ \cdot \}$ denotes the Hamming weight. Since both the parity-check matrix and the error vector are sparse, the probability of $wt \{ s^{(l)} \}$ being significantly different from $w_s^{(l)}$ is negligible.

So, (14) allows predicting the average syndrome weight starting from t'_l . In order to predict how t'_l varies during iterations, let us consider the i -th codeword bit and the corresponding correlation value $\rho_i^{(l)}$ at the l -th iteration. The probability that such a codeword bit is affected by an error can be written as

$$P \left\{ e_i = 1 | \rho_i^{(l)} \right\} = \frac{P \left\{ e_i = 1, \rho_i^{(l)} \right\}}{P \left\{ \rho_i^{(l)} \right\}} = \left(1 + \frac{P \left\{ e_i = 0, \rho_i^{(l)} \right\}}{P \left\{ e_i = 1, \rho_i^{(l)} \right\}} \right)^{-1} \tag{15}$$

where e_i is the i -th bit of the error vector used during encryption. After some calculations, we obtain

$$P \left\{ e_i = 1 | \rho_i^{(l)} \right\} = \frac{1}{1 + \frac{n-t_l}{t_l} \left(\frac{p_{ci}(t_l)}{p_{ic}(t_l)} \right)^{\rho_i^{(l)}} \left(\frac{1-p_{ci}(t_l)}{1-p_{ic}(t_l)} \right)^{m d_v - \rho_i^{(l)}}} \tag{16}$$

where $p_{ci}(t_l)$ and $p_{ic}(t_l)$ are given in (13), with t_l as argument instead of t'_l .

Adding the i -th row of Q^T to the expanded error vector e' is the same as flipping the i -th bit of the error vector e . Hence, we can focus on e and on how its weight t_l changes during decoding iterations. The values of t'_l can be estimated using (14), while, due to sparsity, those of t_l can be estimated as t'_l/m .

The decision to flip the i -th codeword bit is taken when the following condition is fulfilled

$$P \left\{ e_i = 1 | \rho_i^{(l)} \right\} > (1 + \Delta) P \left\{ e_i = 0 | \rho_i^{(l)} \right\} \tag{17}$$

where $\Delta \geq 0$ represents a margin that must be chosen taking into account the DFR and complexity: increasing Δ decreases the DFR but increases the

number of decoding iterations. So, a trade-off value of Δ can be found that allows achieving a low DFR while avoiding unnecessary large numbers of iterations.

Since $P\{e_i = 0|\rho_i^{(l)}\} = 1 - P\{e_i = 1|\rho_i^{(l)}\}$, (17) can be rewritten as

$$P\{e_i = 1|\rho_i^{(l)}\} > \frac{1 + \Delta}{2 + \Delta}. \quad (18)$$

$P\{e_i = 1|\rho_i^{(l)}\}$ is an increasing function of $\rho_i^{(l)}$, hence the minimum value of $\rho_i^{(l)}$ such that (18) is satisfied can be computed as

$$b^{(l)} = \min \left\{ \rho_i^{(l)} \in [0, md_v], \quad \text{s.t.} \quad P\{e_i = 1|\rho_i^{(l)}\} > \frac{1 + \Delta}{2 + \Delta} \right\} \quad (19)$$

and used as the decision threshold at iteration l .

Based on the above considerations, the procedure to compute the decision threshold value per each iteration as a function of the syndrome weight can be summarized as follows:

- i. The syndrome weights corresponding to $t'_l = 0, m, 2m, \dots, mt$ (which are all the possible values of t'_l neglecting cancellations) are computed according to (14). These values are denoted as $\{w_s(0), w_s(m), \dots, w_s(mt)\}$.
- ii. At iteration l , given the syndrome weight $\bar{w}_s^{(l)}$, the integer $j \in [0, t]$ such that $w_s(jm)$ is as close as possible to $\bar{w}_s^{(l)}$ is computed.
- iii. Consider $t_l = j$ and compute $b^{(l)}$ according to (19) and (16). The value of $b^{(l)}$, so obtained, is used as the decoding threshold for iteration l .

The above procedure can be implemented efficiently by populating a look-up table with the pairs $\{w_j, b_j\}$, sequentially ordered. During an iteration, it is enough to compute $\bar{w}_s^{(l)}$, search the largest w_j in the look-up table such that $w_j < \bar{w}_s^{(l)}$ and set $b^{(l)} = b_j$.

We have observed that, moving from large to small values of w_j , the thresholds computed this way firstly exhibit a decreasing trend, then start to increase. According to numerical simulations, neglecting the final increase is beneficial from the performance standpoint. Therefore, in the look-up table we replace the threshold values after the minimum with a constant value equal to the minimum itself.

2.5 Relations with QC-MDPC Code-Based Systems

In LEDAkem, the public code is a QC-MDPC code that admits $L = HQ$ as a valid parity-check matrix. However, differently from QC-MDPC code-based schemes, the private code is a QC-LDPC code, which facilitates decoding. In fact, decoding directly the public QC-MDPC code through classical BF decoders would be a possibility, but the approach we follow is different. By using the decoding algorithm described in Sect. 2.3, we decode the private QC-LDPC code, taking into account the correlation introduced in the private error vector due to

multiplication by Q^T . Since the private QC-LDPC matrix is sparser than the QC-MDPC matrix of the public code, this yields lower decoding complexity.

Besides working over different matrices, the main difference between these two decoding algorithms is in the use of integer multiplications in our decoder, while all multiplications are performed over \mathbb{F}_2 in classical BF decoders. In fact, in our decoder we perform the following operation to compute $R^{(l)}$

$$R^{(l)} = s^{(l-1)} * H * Q = eQ^T H^T * H * Q \approx eL^T * L \quad (20)$$

where the last approximation comes from the fact that, for two sparse matrices A and B , we have $A \cdot B \approx A * B$. Thus, we can say that $HQ \approx H * Q$. So, if we consider classical BF decoding working over the matrix $L = HQ$, the counter vector is computed as

$$\Sigma^{(l)} = s^{(l-1)} * L = eL^T * L. \quad (21)$$

In the Q-decoder, the error vector is updated by summing rows of Q^T , which is equivalent to flipping bits of the public error vector. Hence, there is a clear analogy between decoding of the private QC-LDPC code through the Q-decoder and decoding of the public QC-MDPC code through a classical BF decoder. Through numerical simulations we have verified that the two approaches yield comparable performance in the waterfall region. Performance in the error floor region is instead dominated by the minimum distance of the code over which decoding is performed. Since QC-LDPC codes have smaller minimum distance than QC-MDPC codes, this reflects into a higher error floor when decoding is performed over the private QC-LDPC code. However, no error floor has been observed during simulations of LEDAkem with QC-LDPC decoding, down to a DFR between 10^{-9} and 10^{-8} . Since this is the working point of the codes we use, in terms of DFR, we can say that the error floor effect, if present, is negligible from our scheme performance standpoint.

3 Security Analysis

LEDAkem is constructed starting from the computational problem of syndrome decoding, i.e., obtaining a bounded weight error vector from a given syndrome and a general linear code, which was shown to be NP-complete in [11]. The main difference from the statement of the general hard problem on which our proposal is built is the nature of the code employed, which is quasi-cyclic and admits a representation with a low-density parity-check matrix. To best of our knowledge, there is no superpolynomial advantage in performing syndrome decoding on QC-LDPC, given our public code representation, either due to the quasi-cyclic form of the code or to the low density of its parity matrix. We point out that the same assumption on the lack of advantage due to the quasi-cyclic structure of a code has also been done in both the BIKE [1] and the BIG QUAKE [8] proposals. With these statements standing, the security analysis of LEDAkem examines and quantifies the effectiveness of the best known attacks detailing the efficiency

of algorithms running on both classical and quantum computers providing non-exponential speedups over an enumerative search for the correct error vector. We remark that currently no algorithm running on either a classical Turing Machine (TM) or a quantum TM provides an exponential speedup in solving the computational problem underlying LEDAkem compared to an exhaustive search approach.

3.1 Analysis of the Algorithm with Respect to Known Attacks

As mentioned in the previous sections, LEDAkem derives from QC-LDPC code-based cryptosystems already established in the literature [4, 6]. As proved in [16], in case of using long-term keys, these cryptosystems may be subject to reaction attacks that are able to recover the secret key by exploiting the inherent non-zero DFR they exhibit and Bob's reactions upon decryption failures. However, using ephemeral keys prevents the possibility to mount an attack of this kind, which requires long statistical evaluations. Nevertheless, the risks in case of an accidental keypair reuse must be considered, and this will be done in Sect. 4.

A first type of attacks that can be mounted against LEDAkem are decoding attacks (DAs) aimed at performing decoding through the public code representation, without knowing the private code representation. The most powerful algorithms that can be used for this purpose are ISD algorithms. These algorithms aim at performing decoding of any linear block code by exploiting a general representation of it. ISD algorithms have been introduced by Prange [37] and subsequently improved by Lee-Brickell [23], Leon [24] and Stern [42]. More recently, they have known great advances through modern approaches, also exploiting the generalized birthday paradox [9, 12, 27, 33, 36]. It is possible to show that the general decoding problem is equivalent to the problem of finding low-weight codewords in a general (random-like) code. Therefore, algorithms for searching low-weight codewords can be used as ISD algorithms.

The availability of an efficient algorithm to search for low-weight codewords is also at the basis of key recovery attacks (KRAs). In LEDAkem the matrix $L = HQ$ is a valid parity-check matrix for the public code. Since L is sparse, by knowing it an attacker could separate H from Q and recover the secret key. In order to discover L , an attacker must search for its rows in the dual of the public code. Due to the sparsity of H and Q , any of these rows has weight in the order of $n_0 d_r m$. The attack can be implemented by exploiting again an efficient algorithm for the search of low-weight codewords in linear block codes.

Another potential attack to systems based on QC-LDPC codes is that presented in [40]. This attack uses a special squaring technique and, by extracting the low-weight error vectors, finds low-weight codewords more efficiently than with a general ISD algorithm. This attack, however, is applicable if and only if p is even. Therefore, in order to increase the system security it is advisable to choose odd values of p . Choosing p as a prime is an even more conservative choice against cryptanalysis exploiting factorization of p . The value of p in LEDAkem is chosen in such a way to prevent these attacks.

To estimate complexity of DAs and KRAs exploiting ISD and low-weight codeword searching algorithms, let us define the work factor (WF) of an algorithm as the base-2 logarithm of the average number of binary operations it requires to complete its execution successfully. Let $WF(n, k, w)$ denote the WF of the most efficient algorithm searching for codewords of weight w in a code having length n and dimension k . Such an algorithm can be used to perform ISD with the aim of decrypting a LEDAkem ciphertext without knowing the private key. In this case, we have $n = n_0p$, $k = (n_0 - 1)p$ and $w = t$. Moreover, due to the QC nature of the codes, a speedup in the order of \sqrt{p} must be taken into account [39]. Hence, the security level against decoding attacks of this type can be computed as

$$SL_{DA} = \frac{WF(n_0p, (n_0 - 1)p, t)}{\sqrt{p}}. \quad (22)$$

Concerning the KRAs attack, based on the above considerations we have a similar formula, but with different parameters, that is,

$$SL_{KRA} = \frac{WF(n_0p, p, n_0d_v m)}{p}, \quad (23)$$

where the speedup factor p is due to the fact that recovering only one out of p sparse rows of L , is enough for the attacker (due to the QC structure of L).

According to [43], the most efficient ISD algorithm taking into account Grover’s algorithm [19] running on a quantum computer is Stern’s algorithm. Therefore, the post-quantum security levels have been estimated by considering the work factor of Stern’s algorithm with quantum speedup according to [43]. Instead, with classical computers the most efficient ISD algorithm turns out to be the BJMM algorithm in [9]. Therefore, the security levels against attackers provided with classical computers have been estimated by considering the work factor of BJMM in (22) and (23). We chose to employ the results provided in [43] to evaluate the computational efforts of Stern’s variant of the ISD as they provide exact formulas instead of asymptotic bounds. However, we note that a recent work [22] provides improved asymptotic bounds on the computational complexity of quantum ISD for increasing values of the codeword length n . Deriving from this approach exact values for given parameters set is worth investigating.

3.2 System Parameters

The NIST call for Post-Quantum Cryptography Standardization [32] defines 5 security categories, numbered from 1 to 5 and characterized by increasing strength (see [32] for details). According to this classification, nine instances of LEDAkem are proposed, grouped in three classes corresponding to different security levels. The three instances in each class correspond to three values of n_0 (2, 3, 4), each one yielding a different balance between performance and public key size. The parameters of the nine instances of LEDAkem are reported in Table 1 for the security categories 1, 3 and 5, respectively. In the table, the superscript (pq) denotes that the attack work factor has been computed taking

into account quantum speedups due to Grover’s algorithm, while the superscript (cl) denotes that only classical computers have been considered.

For each security category and considered value of n_0 , we have fixed a value of the parity-check matrix row/column weight d_v in the order of 25 or less (that is advisable to have good error correcting capability of the private QC-LDPC code), and we have found the values of p and m that allow satisfying (23) for the target security level. In fact, the value of m must be chosen such that the dual of the public code, having minimum distance equal to $n_0 m d_v$, is robust against KRAs based on ISD. Once n_0 is fixed, we can find many pairs of values m and d_v which satisfy this bound; among them, we have chosen the one having the lowest product $m d_v$, which is a metric affecting the error correcting capability of the private code. Then, we have found the value of t that allows satisfying (22) and checked whether $t' = tm$ errors can be corrected by the private code through Q-decoding with a sufficiently low DFR. Otherwise, we have increased the value of p keeping all the other parameters fixed. Concerning the estimation of the DFR, we have first exploited BF asymptotic thresholds [6], and then we have performed Montecarlo simulations for each system instance in order to evaluate its DFR. In all Montecarlo simulations, except the one for the Category 1, $n_0 = 2$ parameter set, we have encountered no errors, so the DFR can be approximately bounded by the reciprocal of the number of simulated decryptions. Concerning the parameter set for Category 1, $n_0 = 2$, we obtained 20 failures on $2.394 \cdot 10^9$ decoding computations, pointing to a $\text{DFR} \approx 8.3 \cdot 10^{-9}$.

Table 1. Parameters for LEDAkem and estimated computational efforts to break a given instance as a function of the security category and number of circulant blocks n_0

Category	n_0	p	d_v	$[m_0, \dots, m_{n_0-1}]$	t	$\text{SL}_{\text{DA}}^{(\text{pq})}$	$\text{SL}_{\text{KRA}}^{(\text{pq})}$	$\text{SL}_{\text{DA}}^{(\text{cl})}$	$\text{SL}_{\text{KRA}}^{(\text{cl})}$	DFR
1	2	27,779	17	[4, 3]	224	135.43	134.84	217.45	223.66	$\approx 8.3 \cdot 10^{-9}$
	3	18,701	19	[3, 2, 2]	141	135.63	133.06	216.42	219.84	$\lesssim 10^{-9}$
	4	17,027	21	[4, 1, 1, 1]	112	136.11	139.29	216.86	230.61	$\lesssim 10^{-9}$
2–3	2	57,557	17	[6, 5]	349	200.47	204.84	341.52	358.16	$\lesssim 10^{-8}$
	3	41,507	19	[3, 4, 4]	220	200.44	200.95	341.61	351.57	$\lesssim 10^{-8}$
	4	35,027	17	[4, 3, 3, 3]	175	200.41	201.40	343.36	351.96	$\lesssim 10^{-8}$
4–5	2	99,053	19	[7, 6]	474	265.38	267.00	467.24	478.67	$\lesssim 10^{-8}$
	3	72,019	19	[7, 4, 4]	301	265.70	270.18	471.67	484.48	$\lesssim 10^{-8}$
	4	60,509	23	[4, 3, 3, 3]	239	265.48	268.03	473.38	480.73	$\lesssim 10^{-8}$

In order to make a conservative design of the system, we have considered some margin in the complexity estimates of the attacks, such that the actual security level for these instances is larger than the target one. This also accounts for possible (though rare) cancellations occurring in L , which may yield a row weight

slightly smaller than $md_v n_0$. The values of d_v have been chosen greater than 15 in order to avoid codes having too small minimum distances. In addition, they are odd to ensure that the circulant blocks forming H and L (and L_{n_0-1} , in particular) have full rank. Also the values of m are always odd, and the sets $[m_0, m_1, \dots, m_{n_0-1}]$ have been chosen in such a way to guarantee that Q has full rank. In fact, $L = HQ$ is a valid parity-check matrix for the public code: if Q is singular, it might happen that the rank of L is lower than p , leading to a code with a co-dimension lower than p . With the choice of an invertible Q , we guarantee that this does not occur.

4 Properties of the Proposed Cryptosystem

The QC-LDPC code-based Niederreiter cryptosystem alone achieves only indistinguishability under chosen plaintext attack (IND-CPA), that however is sufficient in case of using ephemeral keys. It is possible to convert a Niederreiter cryptosystem achieving only IND-CPA into one achieving indistinguishability under chosen ciphertext attack (IND-CCA), under the assumption that the DFR of the underlying code is zero. Such a conversion involves substituting the outcome of a decoding failure (due to an ill-formed ciphertext) with the outcome of a KDF taking as input either the public syndrome and a fixed secret bit sequence [21, 38], or a secret permutation of the syndrome itself [13]. We apply the conversion specified in [21] to our scheme, despite its DFR is not null, as it still proves beneficial in case of an accidental keypair reuse, against an attacker matching the IND-CCA model whenever no decoding failures due to the QC-LDPC code structure takes place. Furthermore, we note that LEDAkem ciphertexts are not malleable in a chosen plaintext scenario. Indeed, even if an attacker alters arbitrarily a ciphertext so that it decrypts to a valid error vector e (e.g., discarding the ciphertext and forging a new one), the shared secret is derived via a hash based KDF, which prevents him from controlling the output of the decryption.

Relations with the Security of QC-MDPC Code-Based Systems. Differently from QC-MDPC code-based systems, the public code in LEDAkem has a QC-MDPC matrix L that can be factorized into H and Q , and this might appear to yielding lower security than a general QC-MDPC matrix. However, in order to attempt factorization of L , the attacker should first recover it by searching for low-weight codewords in the dual of the public code. Once L has been recovered, trying to factorize it into H and Q indeed becomes pointless, since the attacker could exploit L to perform direct decoding of the public QC-MDPC code. Alternatively, an attacker could try to perform decoding of the public code, which requires solving the syndrome decoding problem for the same code. The best known techniques for solving these two problems are based on ISD, and no method is known to facilitate their solution by exploiting the fact that L can be factorized into H and Q .

Risks in Case of Keypair Reuse. While LEDAkem uses ephemeral keys that are meant for single use, it is possible that implementation accidents lead to a reuse of the same keypair more than once. The main threat in case of keypair reuse is the reaction attack described in [16], where a correlation between the DFR and the private key is derived. However, for the attack to succeed, the attacker needs to reliably estimate the decoding failure rate for a set of carefully crafted or selected error vectors. Given the DFR for which LEDAkem was designed ($<10^{-8}$), obtaining a reliable estimate requires a number of decryptions with the same key in the order of billions. Since the said evaluation should be obtained for all the possible distances between two set bits in the secret key, a conservative estimate of the number of decryption actions required is $(p-1)\frac{1}{DFR}$, which, considering the weakest case, corresponding to Category 1 with $n_0 = 2$, yields $\gtrsim 2.7 \times 10^{12}$ decryptions. Therefore, the attack presented in [16] is not a practical threat on LEDAkem with the proposed parameters, unless a significant amount of decryptions are performed with the same key. Moreover, even the chosen ciphertext attack (CCA) described in [13], where a ciphertext is crafted with a number of errors greater than t to artificially increase the DFR of the system, can be thwarted through checking the weight of the decoded error vector and reporting a decoding failure if it exceeds t .

Protection Against Side-Channel Attacks. The two most common side channels exploited to breach practical implementations of cryptosystems are the execution time of the primitive and the instantaneous power consumption during its computation. In particular, in [15], it was shown how a QC-LDPC code-based system can be broken by means of simple power analysis, exploiting the control-flow dependent differences of the decoding algorithm. We note that employing ephemeral keys provides a natural resistance against non-profiled power consumption side channel attacks, as a significant amount of measurements with the same key (>30) must be collected before the key is revealed.

Concerning execution time side channel information leakage, the main portion of the LEDAkem decryption algorithm which is not characterized by a constant execution time is decoding. Indeed, the number of iterations made by the decoder depends on the values being processed. However, for the proposed parameters, we note that the number of iterations is between 3 and 5, with a significant bias towards 4. Hence, it is simple to achieve a constant time decoding by modifying the algorithm so that it always runs for the maximum needed amount of iterations to achieve the desired DFR. Such a choice completely eliminates the timing leakage, albeit trading it off for a performance penalty.

5 Implementation and Numerical Results

An effort has been made to realize a fast and efficient C99 implementation of LEDAkem without platform-dependent optimizations, which is publicly available in [5]. To this end, we represented each circulant block as a polynomial in $\mathbb{F}_2[x]/\langle x^p+1 \rangle$ thanks to the isomorphism described in Sect. 2.1. Consequently, all

Table 2. Running times for key generation, encryption and decryption as a function of the category and the number of circulant blocks n_0 on an AMD Ryzen 5 1600 CPU.

Category	n_0	KeyGen (ms)	Encrypt (ms)	Decrypt (ms)	Total CPU time ephemeral KEM (ms)
1	2	34.11 (± 1.07)	2.11 (± 0.08)	16.78 (± 0.53)	52.99
	3	16.02 (± 0.26)	2.15 (± 0.17)	21.65 (± 1.71)	39.81
	4	13.41 (± 0.23)	2.42 (± 0.08)	24.31 (± 0.86)	40.14
2–3	2	142.71 (± 1.52)	8.11 (± 0.21)	48.23 (± 2.93)	199.05
	3	76.74 (± 0.78)	8.79 (± 0.20)	49.15 (± 2.20)	134.68
	4	54.93 (± 0.84)	9.46 (± 0.28)	46.16 (± 2.03)	110.55
4–5	2	427.38 (± 5.15)	23.00 (± 0.33)	91.78 (± 5.38)	542.16
	3	227.71 (± 1.71)	24.85 (± 0.37)	92.42 (± 4.50)	344.99
	4	162.34 (± 2.39)	26.30 (± 0.53)	127.16 (± 4.42)	315.80

the involved block circulant matrices are represented as matrices of polynomials in $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$. The polynomials are materialized employing a bit-packed form of their binary coefficients in all the cases where the number of non-null coefficients is high. In case a polynomial has a low number of non-null coefficients with respect to the maximum possible, i.e., the circulant matrix is sparse, we materialize only the positions of its one coefficients as integers.

We provide below the results of a set of execution time benchmarks. The results were obtained measuring the required time for key generation, encryption (key encapsulation) and decryption (key decapsulation) as a function of the chosen security category and the number of circulant blocks n_0 . The measurements reported are obtained as the average of 100 executions of the reference implementation. The generated binaries were run on an AMD Ryzen 5 1600 CPU at 3.2 GHz, locking the frequency scaling to the top frequency.

Table 2 reports the running times in terms of CPU time taken by the process. As it can be noticed, the most computationally demanding primitive is the key generation, which has more than 80% of its computation time taken by the execution of a single modular inverse in $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$ required to obtain the value of $L_{n_0-1}^{-1}$. The encryption primitive is the fastest among all, and its computation time is substantially entirely devoted ($>99\%$) to the $n_0 - 1$ polynomial multiplications performing the encryption. The decryption primitive computation is dominated by the Q-decoder computation ($>95\%$ of the time), with a minimal portion taken by the n_0 modular multiplications which reconstruct L_{n_0-1} and the one to compute the private syndrome fed into the Q-decoder.

Considering the computational cost of performing a KEM with ephemeral keys, the most advantageous choice is to pick $n_0 = 4$ for any security level, although the computational savings are more significant when considering high-security parameter choices (Category 3 and 5).

Table 3. Sizes of the keypair and encapsulated shared secret as a function of the chosen category and number of circulant blocks n_0 .

Category	n_0	Private key size (B)		Public key size (B)	Shared secret size (B)	Enc secret size (B)
		At rest	In memory			
1	2	24	668	3,480	3,480	32
	3	24	844	4,688	2,344	32
	4	24	1,036	6,408	2,136	32
2-3	2	32	972	7,200	7,200	48
	3	32	1,196	10,384	5,192	48
	4	32	1,364	13,152	4,384	48
4-5	2	40	1,244	12,384	12,384	64
	3	40	1,548	18,016	9,008	64
	4	40	1,772	22,704	7,568	64

Table 3 reports the sizes of both the keypairs and the encapsulated secrets for LEDAkem. In particular, regarding the size of the private keys we report both the size of the stored private key and the required amount of main memory to store the expanded key during the decryption phase. We note that, for a given security category, increasing the value of n_0 enlarges the public key, as it is constituted of $(n_0 - 1)p$ bits. This increase in the size of the public key represents a tradeoff with the decrease of the size of the ciphertext to be transmitted since it is only p bits long, and p decreases if a larger number of blocks is selected, for a fixed security category. The size of the derived encapsulated secret is at least 256 bits, in order to meet the requirement reported in [32]. The shared secret is derived employing the SHA-3 hash function with a 256, 384 or 512 bits digest, in order to match the requirements of Categories 1, 3, and 5, respectively.

6 Conclusion

We have introduced a post-quantum KEM based on QC-LDPC codes with the following advantages: it is built on an NP-complete problem under reasonable assumptions; it exploits improved BF decoders which are faster than classical BF decoders; it requires compact keypairs (below 23 kiB at most), with minimum size private keys; it needs only addition and multiplication over $\mathbb{F}_2[x]$, and modular inverse over $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$ besides single-precision integer operations; it is particularly efficient in applying countermeasures against non-profiled power consumption side channel attacks. As regards implementation, no platform specific optimizations have been exploited, thus we expect these results to be quite consistent across different platforms. On the other hand, starting from this platform-agnostic reference implementation, a number of optimizations can be applied to make LEDAkem faster.

Acknowledgments. Paolo Santini was partly funded by Namirial SpA.

References

1. Aragon, N., Barreto, P.S.L.M., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Güneysu, T., Melchor, C.A., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.P., Zémor, G.: BIKE: bit flipping key encapsulation (2017). <http://bikesuite.org/files/BIKE.pdf>
2. Baldi, M., Bianchi, M., Chiaraluca, F.: Optimization of the parity-check matrix density in QC-LDPC code-based McEliece cryptosystems. In: Proceedings of the IEEE ICC 2013 - Workshop on Information Security over Noisy and Lossy Communication Systems, Budapest, Hungary, June 2013
3. Baldi, M., Bodrato, M., Chiaraluca, F.: A new analysis of the McEliece cryptosystem based on QC-LDPC codes. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 246–262. Springer, Heidelberg (2008). <https://doi.org/10.1007/978-3-540-85855-3-17>
4. Baldi, M.: QC-LDPC Code-Based Cryptography. SpringerBriefs in Electrical and Computer Engineering. Springer, Cham (2014). <https://doi.org/10.1007/978-3-319-02556-8>
5. Baldi, M., Barengi, A., Chiaraluca, F., Pelosi, G., Santini, P.: LEDAkem: Low dEnsity coDe-bAsed key encapsulation mechanism (2017). <https://www.ledacrypt.org/>
6. Baldi, M., Bianchi, M., Chiaraluca, F.: Security and complexity of the McEliece cryptosystem based on QC-LDPC codes. IET Inf. Secur. **7**(3), 212–220 (2012)
7. Baldi, M., Bianchi, M., Chiaraluca, F., Rosenthal, J., Schipani, D.: Enhanced public key security for the McEliece cryptosystem. J. Cryptol. **29**(1), 1–27 (2016)
8. Bardet, M., Barelli, E., Blazy, O., Torres, R.C., Couvreur, A., Gaborit, P., Otmani, A., Sendrier, N., Tillich, J.P.: BIG QUAKE: BINARY Goppa QUASI-cyclic Key Encapsulation (2017). <https://bigquake.inria.fr/files/2017/12/proposal.pdf>
9. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: how $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_31
10. Berger, T.P., Loidreau, P.: How to mask the structure of codes for a cryptographic use. Des. Codes Crypt. **35**(1), 63–79 (2005)
11. Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Trans. Inf. Theory **24**(3), 384–386 (1978)
12. Bernstein, D.J., Lange, T., Peters, C.: Smaller decoding exponents: ball-collision decoding. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 743–760. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_42
13. Cayrel, P.-L., Gueye, C.T., Mboup, E.H.M., Ndiaye, O., Persichetti, E.: Efficient implementation of hybrid encryption from coding theory. In: El Hajji, S., Nitaj, A., Souidi, E.M. (eds.) C2SI 2017. LNCS, vol. 10194, pp. 254–264. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55589-8_17
14. Chaulet, J., Sendrier, N.: Worst case QC-MDPC decoder for McEliece cryptosystem. In: Proceedings of the IEEE International Symposium on Information Theory (ISIT 2016), Barcelona, Spain, pp. 1366–1370, July 2016
15. Fabšič, T., Gallo, O., Hromada, V.: Simple power analysis attack on the QC-LDPC McEliece cryptosystem. Tatra Mt. Math. Pub. **67**(1), 85–92 (2016)

16. Fabšič, T., Hromada, V., Stankovski, P., Zajac, P., Guo, Q., Johansson, T.: A reaction attack on the QC-LDPC McEliece cryptosystem. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 51–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_4
17. Gaborit, P.: Shorter keys for code based cryptography. In: Proceedings of the International Workshop on Coding and Cryptography (WCC 2005), Bergen, Norway, pp. 81–90, March 2005
18. Goppa, V.D.: A new class of linear correcting codes. Probl. Pered. Inform. **6**(3), 24–30 (1970)
19. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, Philadelphia, PA, pp. 212–219, May 1996
20. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_29
21. Hofheinz, D., Hvelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. Cryptology ePrint Archive, Report 2017/604 (2017). <https://eprint.iacr.org/2017/604>
22. Kachigar, G., Tillich, J.-P.: Quantum information set decoding algorithms. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 69–89. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_5
23. Lee, P.J., Brickell, E.F.: An observation on the security of McEliece’s public-key cryptosystem. In: Barstow, D., et al. (eds.) EUROCRYPT 1988. LNCS, vol. 330, pp. 275–280. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-45961-8_25
24. Leon, J.: A probabilistic algorithm for computing minimum weights of large error-correcting codes. IEEE Trans. Inf. Theory **34**(5), 1354–1359 (1988)
25. Li, Y.X., Deng, R., Wang, X.M.: On the equivalence of McEliece’s and Niederreiter’s public-key cryptosystems. IEEE Trans. Inf. Theory **40**(1), 271–273 (1994)
26. Löndahl, C., Johansson, T.: A new version of McEliece PKC based on convolutional codes. In: Chim, T.W., Yuen, T.H. (eds.) ICICS 2012. LNCS, vol. 7618, pp. 461–470. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34129-8_45
27. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_6
28. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. DSN Progress Report, pp. 114–116 (1978)
29. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: new McEliece variants from moderate density parity-check codes. In: 2013 IEEE International Symposium on Information Theory, pp. 2069–2073, July 2013
30. 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
31. Monico, C., Rosenthal, J., Shokrollahi, A.: Using low density parity check codes in the McEliece cryptosystem. In: Proceedings of the IEEE International Symposium on Information Theory (ISIT 2000), Sorrento, Italy, p. 215, June 2000
32. National Institute of Standards and Technology: Post-quantum crypto project, December 2016. <http://csrc.nist.gov/groups/ST/post-quantum-crypto/>

33. Niebuhr, R., Persichetti, E., Cayrel, P.L., Bulygin, S., Buchmann, J.: On lower bounds for information set decoding over f_q and on the effect of partial knowledge. *Int. J. Inf. Coding Theory* **4**(1), 47–78 (2017)
34. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. *Probl. Control Inf. Theory* **15**, 159–166 (1986)
35. Otmani, A., Tillich, J.P., Dallot, L.: Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes. In: *Proceedings of the First International Conference on Symbolic Computation and Cryptography (SCC 2008)*, Beijing, China, April 2008
36. Peters, C.: Information-set decoding for linear codes over \mathbf{F}_q . In: Sendrier, N. (ed.) *PQCrypto 2010*. LNCS, vol. 6061, pp. 81–94. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12929-2_7
37. Prange, E.: The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory* **8**(5), 5–9 (1962)
38. Saito, T., Xagawa, K., Yamakawa, T.: Tightly-secure key-encapsulation mechanism in the quantum random oracle model. *Cryptology ePrint Archive*, Report 2017/1005 (2017). <https://eprint.iacr.org/2017/1005>
39. Sendrier, N.: Decoding one out of many. In: Yang, B.-Y. (ed.) *PQCrypto 2011*. LNCS, vol. 7071, pp. 51–67. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_4
40. Shooshtari, M.K., Ahmadian-Attari, M., Johansson, T., Aref, M.R.: Cryptanalysis of McEliece cryptosystem variants based on quasi-cyclic low-density parity check codes. *IET Inf. Secur.* **10**(4), 194–202 (2016)
41. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**(5), 1484–1509 (1997)
42. Stern, J.: A method for finding codewords of small weight. In: Cohen, G., Wolfmann, J. (eds.) *Coding Theory 1988*. LNCS, vol. 388, pp. 106–113. Springer, Heidelberg (1989). <https://doi.org/10.1007/BFb0019850>
43. de Vries, S.: Achieving 128-bit security against quantum attacks in OpenVPN. Master’s thesis, University of Twente, August 2016. <http://essay.utwente.nl/70677/>



Decoding Linear Codes with High Error Rate and Its Impact for LPN Security

Leif Both^(✉) and Alexander May

Faculty of Mathematics, Horst Görtz Institute for IT-Security,
Ruhr-University Bochum, Bochum, Germany
{leif.both,alex.may}@rub.de

Abstract. We propose a new algorithm for the decoding of random binary linear codes of dimension n that is superior to previous algorithms for high error rates. In the case of Full Distance decoding, the best known bound of $2^{0.0953n}$ is currently achieved via the BJMM-algorithm of Becker, Joux, May and Meurer. Our algorithm significantly improves this bound down to $2^{0.0885n}$.

Technically, our improvement comes from the heavy use of Nearest Neighbor techniques in all steps of the construction, whereas the BJMM-algorithm can only take advantage of Nearest Neighbor search in the last step.

Since cryptographic instances of LPN usually work in the high error regime, our algorithm has implications for LPN security.

Keywords: Decoding binary linear codes · BJMM
Nearest Neighbors · LPN · Full Distance decoding · Representations

1 Introduction

The NP-hard decoding problem for random linear codes plays a major role in coding and complexity theory. It is especially suitable for the construction of quantum-secure cryptographic systems like [McE78, Ale03, Reg05]. In view of the upcoming NIST selection of post-quantum public-key cryptosystems [NIS] it is of crucial importance for secure parameter selection to know the best decoding algorithms.

A linear code \mathcal{C} is a k -dimensional subspace of \mathbb{F}_2^n . In the decoding problem the attacker gets an erroneous version $\mathbf{x} = \mathbf{c} + \mathbf{e}$ of a codeword \mathbf{c} for some error vector \mathbf{e} with Hamming weight $\Delta(\mathbf{e}) = \omega$. His target is to find \mathbf{e} in order to recover the original codeword \mathbf{c} . Sometimes, the weight ω is bounded by the distance d of the code \mathcal{C} (Full Distance Decoding) or by $\frac{d}{2}$ (Half Distance Decoding).

Therefore the running time $T(n, k, d)$ of any decoding algorithm is a function of the parameters n , k and d . It is well known that the Gilbert-Varshamov bound gives us $\frac{k}{n} \approx 1 - H\left(\frac{d}{n}\right)$ for random linear codes, where $H(\cdot)$ is the binary entropy function $H(p) := -p \log(p) - (1-p) \log(1-p)$. This results in a running time

$T(n, k)$ which is a function of n and k only. Furthermore one often compares worst case running times where we maximize the running time over all rates $\frac{k}{n}$ resulting in a running time $T(n)$.

The best algorithmic paradigm that we know today for random binary linear codes is a class of algorithms called *Information Set Decoding* (ISD). Here, for simplicity we only compare ISD running times in the Full Distance Decoding setting, but see also Fig. 1. For all ISD algorithms the maximal run time is achieved at a rate $\frac{k}{n}$ slightly below $\frac{1}{2}$.

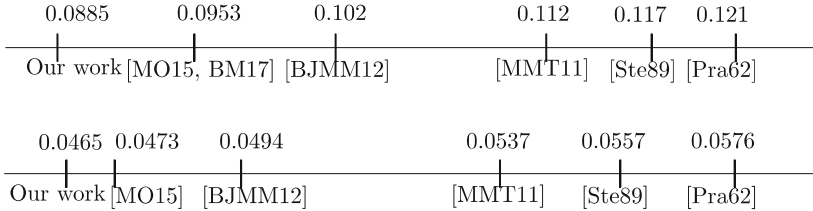


Fig. 1. Comparison for Full/Half Distance of our work and other algorithms.

The first ISD algorithm is due to Prange [Pra62] and achieves worst case running time $2^{0.121n}$. This was improved by Stern and Dumer [Ste88, Dum91] to $2^{0.117n}$. Using the representation technique, May et al. [MMT11] and later Becker et al. [BJMM12] further decreased the run time to $2^{0.112n}$ and $2^{0.102n}$, respectively. The last is called BJMM algorithm and is currently asymptotically the best algorithm for decoding of random linear codes.

In 2015, May and Ozerov [MO15] proposed some Nearest Neighbor (NN) search that further sped up BJMM to $2^{0.0967n}$, which was later optimized in [BM17b] to $2^{0.0953n}$.

Our results. As can be seen from Fig. 1, our new algorithm achieves in the Full Distance Decoding setting $2^{0.0885n}$, which is a quite remarkable improvement over the current state of the art. However, the improvement for the Half Distance Decoding is comparably small. As a rule of thumb, the larger the error rate, the more significant our algorithm’s improvement.

As most promising in cryptographic settings, we currently see the application of our algorithm for Learning Parity with Noise (LPN) instances. Every LPN instance of dimension k with error τ is naturally a decoding problem for a random linear code. As shown by Esser et al. [EKM17] in practice one currently best solves large LPN instances by a hybrid approach. Namely, one first applies a dimension reduction algorithm (such as BKW [GJL14]) at the cost of introducing a large error close to $\frac{1}{2}$, followed by a decoding algorithm. Since our algorithm works especially well in the high-error regime, it seems to be a perfect candidate for solving these transformed LPN instances.

Our algorithm. ISD algorithms with representation technique such as MMT and BJMM currently use a 2-step matching process, where in the first step one does an exact matching of vectors (for eliminating representations) and in a

second step one does an approximate matching via NN search. We eliminate this two-step process and perform only an approximate matching in all stages of the algorithm.

This allows us to eliminate representations less restrictive, and to use the full power of NN search in every step of our algorithm. Thus, our approximate matching is in spirit similar to the Ball Decoding approach of Bernstein et al. [BLP11]. The heavy use of NN search might also explain the large improvement (only) in the high error-regime, where NN search can show its full strength.

This paper is organized as follows. In Sect. 2, we review some ISD algorithms. Section 3 introduces a basic version of our new algorithm, whereas the generalized version is given in Sect. 4. Our results are provided in Sect. 5.

2 Preliminaries

Syndrome Decoding. Let us start with some preliminaries on linear codes and decoding algorithms. We denote the *Hamming distance* of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$ by $\Delta(\mathbf{x}, \mathbf{y})$. The Hamming weight $\Delta(\mathbf{x})$ of \mathbf{x} is defined as the Hamming distance of \mathbf{x} to the zero vector $\mathbf{0}$.

A linear code \mathcal{C} is a k -dimensional subspace of \mathbb{F}_2^n . Its distance is defined by $d := \min_{\mathbf{c} \neq \mathbf{c}' \in \mathcal{C}} \{\Delta(\mathbf{c}, \mathbf{c}')\}$. We can specify \mathcal{C} by a generator matrix $G \in \mathbb{F}_2^{k \times n}$ or a parity check matrix $P \in \mathbb{F}_2^{(n-k) \times n}$ via

$$\mathcal{C} := \{\mathbf{x}G \in \mathbb{F}_2^n \mid \mathbf{x} \in \mathbb{F}_2^k\} \text{ or } \mathcal{C} := \{\mathbf{c} \in \mathbb{F}_2^n \mid P\mathbf{c} = \mathbf{0}\}.$$

Random linear codes have a random G or random P , where in both cases each matrix entry is chosen uniformly at random from \mathbb{F}_2 . For an arbitrary vector $\mathbf{y} = \mathbf{c} + \mathbf{e} \in \mathbb{F}_2^n, \mathbf{c} \in \mathcal{C}$ we define the syndrome of \mathbf{y} as

$$\mathbf{s} := P\mathbf{y} = P\mathbf{c} + P\mathbf{e} = P\mathbf{e}. \quad (1)$$

Definition 1 (Syndrome Decoding Problem). *Let \mathcal{C} be a linear code specified by some parity check matrix $P \in \mathbb{F}_2^{(n-k) \times n}$. Given P , an (erroneous) codeword $\mathbf{y} \in \mathbb{F}_2^n$ and a weight $\omega \in \mathbb{N}$, one has to find an error vector $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathbf{y} + \mathbf{e} \in \mathcal{C}$ and $\Delta(\mathbf{e}) = \omega$.*

We call (P, \mathbf{s}, ω) with $\mathbf{s} = P\mathbf{y}$ an instance of the Syndrome Decoding Problem. We say that $\mathbf{e} \in \mathbb{F}_2^n$ solves (P, \mathbf{s}, ω) iff $\mathbf{s} = P\mathbf{e}$ and $\Delta(\mathbf{e}) = \omega$.

A fundamental algorithm for solving the Syndrome Decoding Problem was introduced by Prange [Pra62]. This algorithm is the basis of all of today's so-called Information Set Decoding (ISD) algorithms [Ste88, Dum91, BLP11, MMT11, BJMM12]. In Prange's algorithm, one reduces the dimension of the search space from n down to k via Gaussian elimination.

In more detail, one chooses some invertible $G \in \mathbb{F}_2^{(n-k) \times (n-k)}$ such that $GP = (\bar{P} \mid I_{n-k})$, where I_{n-k} is the $(n-k)$ -dimensional identity matrix. Therefore Eq. (1) becomes

$$GPe = (\bar{P} \mid I_{n-k})\mathbf{e} = \bar{P}\mathbf{e}' + \mathbf{e}'' = G\mathbf{s} =: \bar{\mathbf{s}}, \text{ with } \mathbf{e} = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}. \quad (2)$$

Thus, every instance (P, \mathbf{s}, ω) with $P \in \mathbb{F}_2^{(n-k) \times n}$ of the Decoding Problem has some (non-unique) standard form $(\bar{P}, \bar{\mathbf{s}}, \omega)$ with $\bar{P} \in \mathbb{F}_2^{(n-k) \times k}$ such that $\mathbf{e} \in \mathbb{F}_2^n$ solves (P, \mathbf{s}, ω) iff $(\bar{P} \mid I_{n-k})\mathbf{e} = \bar{\mathbf{s}}$.

Definition 2 (Standard form). *For any instance $(P, \mathbf{s}, \omega) \in \mathbb{F}_2^{(n-k) \times n} \times \mathbb{F}_2^{n-k} \times \mathbb{N}$ of the decoding problem, we say that $(\bar{P}, \bar{\mathbf{s}}, \omega) \in \mathbb{F}_2^{(n-k) \times k} \times \mathbb{F}_2^{n-k} \times \mathbb{N}$ is a standard form of (P, \mathbf{s}, ω) if there exists some invertible $G \in \mathbb{F}_2^{(n-k) \times (n-k)}$ such that*

$$GP = (\bar{P} \mid I_{n-k}) \text{ and } G\mathbf{s} = \bar{\mathbf{s}}.$$

The underlying idea of all Information Set Decoding algorithm is to solve a dimension-reduced standard form $(\bar{P}, \bar{\mathbf{s}}, \omega)$ of a Decoding Problem instance instead of its original form (P, \mathbf{s}, ω) .

However, before transforming (P, \mathbf{s}, ω) to its normal form one applies some column permutation π to P to enforce a special weight distribution on $\mathbf{e} = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$. While Prange chooses $\Delta(\mathbf{e}') = 0$, other ISD algorithms enforce $\Delta(\mathbf{e}') = p$ for some parameter $p \geq 0$ (see Algorithms 1 and 2). Thus it is sufficient to find some $\mathbf{e}' \in \mathbb{F}_2^k$, $\Delta(\mathbf{e}') = p$ such that after applying π and converting to standard form the term $\bar{P}\mathbf{e}'$ is close to $\bar{\mathbf{s}}$, i.e.,

$$\Delta(\bar{P}\mathbf{e}', \bar{\mathbf{s}}) = \Delta(\mathbf{e}'') = \omega - p.$$

Algorithm 1. ISD – WEIGHT DISTRIBUTION AND STANDARD FORM

Input : $P \in \mathbb{F}_2^{(n-k) \times n}$, $\mathbf{s} \in \mathbb{F}_2^{n-k}$, $\omega \in \mathbb{N}$
Output: $\mathbf{e} \in \mathbb{F}_2^n$ with $P\mathbf{e} = \mathbf{s}$ and $\Delta(\mathbf{e}) = \omega$
repeat
 repeat
 $\pi \leftarrow$ random permutation on \mathbb{F}_2^n
 $(\cdot \mid Q) \leftarrow \pi(P)$ (permute columns) $\triangleright Q \in \mathbb{F}_2^{(n-k) \times (n-k)}$
 until Q is invertible
 $(\bar{P} \mid I_{n-k}) \leftarrow G\pi(P)$ and $\bar{\mathbf{s}} \leftarrow G\mathbf{s}$ $\triangleright G \in \mathbb{F}_2^{(n-k) \times (n-k)}$
 $(\mathbf{e}', \mathbf{e}'') = \text{ISDSOLVE}(\bar{P}, \bar{\mathbf{s}}, \omega)$ \triangleright See Algorithm 2.
until $(\mathbf{e}', \mathbf{e}'') \neq \perp$
return $\pi^{-1}(\mathbf{e}' \parallel \mathbf{e}'')$

Algorithm 2. ISDSOLVE

Input : $\bar{P} \in \mathbb{F}_2^{(n-k) \times k}$, $\bar{\mathbf{s}} \in \mathbb{F}_2^{n-k}$, $\omega \in \mathbb{N}$
Output : $(\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$
Parameters: choose optimal $0 \leq p \leq \omega$
for $\mathbf{e}' \in \mathbb{F}_2^k$ with $\Delta(\mathbf{e}') = p$ **do**
 $\mathbf{e}'' \leftarrow H\mathbf{e}' + \bar{\mathbf{s}}$
 if $\Delta(\mathbf{e}'') = \omega - p$ **then return** $(\mathbf{e}', \mathbf{e}'')$
end
return \perp

Dumer's ISD-algorithm [Dum91] introduces another parameter ℓ and transforms P into a different standard form

$$G'P = \begin{pmatrix} \bar{P}_1 & 0 \\ \bar{P}_2 & I_{n-k-\ell} \end{pmatrix}, \text{ where } \bar{P}_1 \in \mathbb{F}_2^{\ell \times (k+\ell)} \text{ and } \bar{P}_2 \in \mathbb{F}_2^{(n-k-\ell) \times (k+\ell)}.$$

Set $\bar{\mathbf{s}} := G'\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{F}_2^\ell \times \mathbb{F}_2^{n-k-\ell}$. We can now write Eq. (2) as

$$\bar{P}_1 \mathbf{e}_1^{(1)} = \bar{P}_1 \mathbf{e}_2^{(1)} + \mathbf{s}_1 \quad \text{and} \quad (3)$$

$$\Delta(\bar{P}_2 \mathbf{e}_1^{(1)}, \bar{P}_2 \mathbf{e}_2^{(1)} + \mathbf{s}_2) = \omega - p. \quad (4)$$

splitting $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ with $\mathbf{e}', \mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)} \in \mathbb{F}_2^{k+\ell}$. Hence, by Eq. (3) we have an *exact* matching of $\bar{P}\mathbf{e}'$ and $\bar{\mathbf{s}}$ on ℓ coordinates, and by Eq. (4) an *approximate* matching of the same vectors on the remaining $n - k - \ell$ coordinates.

The BJMM algorithm [BJMM12] solves the exact matching of Eq. (3). In a nutshell, BJMM constructs solutions $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ for Eq. (3) using some depth-3 binary search tree. For optimizing the depth of this search tree, see [BM17b]. All candidate solutions $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ are then checked via Eq. (4).

For the approximate matching, May and Ozerov [MO15] proposed a Nearest Neighbor (NN) search algorithm that, given two lists L_1, L_2 , finds in time sub-quadratic of the list lengths all elements $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$ within some given Hamming distance $\Delta(\mathbf{x}_1, \mathbf{x}_2)$. Thus, May-Ozerov NN search can be used to speed up the check of candidate solutions via Eq. (4) inside the BJMM algorithm.

Theorem 1 ([MO15]). *Given two lists L_1, L_2 with elements taken uniformly at random from \mathbb{F}_2^n and length $|L_1|, |L_2| \leq 2^{\lambda n}$. Then for any $\epsilon > 0$ one can find all but a negligible fraction of the pairs $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$ satisfying $\Delta(\mathbf{x}_1, \mathbf{x}_2) \leq \gamma n$ for some given $0 \leq \gamma \leq \frac{n}{2}$ provided that $\lambda < 1 - H(\frac{\gamma}{2})$ in time*

$$2^{y(\lambda, \gamma) + \epsilon n}, \text{ where } y(\lambda, \gamma) := (1 - \gamma) \left(1 - H \left(\frac{H^{-1}(1 - \lambda) - \frac{\gamma}{2}}{1 - \gamma} \right) \right).$$

Please notice that Theorem 1 can only be applied for parameters satisfying the condition $\lambda < 1 - H(\frac{\gamma}{2})$, which will not always be the case for our new decoding algorithm. Whenever this condition is violated, we will choose one of the following two simple NN search algorithms Algorithms 3 or 4.

Algorithm 3. NN-ENUMERATE-PAIRS

Input : $L_1, L_2 \subset \mathbb{F}_2^n, \gamma$
Output: L
for $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$ **do**
 | **if** $\Delta(\mathbf{x}_1, \mathbf{x}_2) \leq \gamma n$ **then** $L \leftarrow (\mathbf{x}_1, \mathbf{x}_2)$
end
return L

Since Algorithm 3 simply tests the distance of all pairs in $L_1 \times L_2$, it runs in time quadratic in the list lengths

$$2^{2\lambda n}. \quad (5)$$

Notice that here, as in the rest of the paper, we neglect for ease of presentation polynomial factors in the run time.

Algorithm 4. NN-MEET-IN-THE-MIDDLE

Input : $L_1, L_2 \subset \mathbb{F}_2^n, \gamma$
Output: L
 $L'_2 \leftarrow \emptyset$
for $\mathbf{x}_2 \in L_2$ **do**
 | **for** $\mathbf{e} \in \mathbb{F}_2^n$ with $\Delta(\mathbf{e}) \leq \frac{\gamma}{2}n$ **do**
 | | $L'_2 \leftarrow L'_2 \cup (\mathbf{x}_2 + \mathbf{e}, \mathbf{x}_2)$
 | **end**
end
for $\mathbf{x}_1 \in L_1$ **do**
 | **for** $\mathbf{e} \in \mathbb{F}_2^n$ with $\Delta(\mathbf{e}) \leq \frac{\gamma}{2}n$ **do**
 | | **if** $(\mathbf{x}_1 + \mathbf{e}, \mathbf{x}_2) \in L'_2$ **then** $L \leftarrow (\mathbf{x}_1, \mathbf{x}_2)$
 | **end**
end
return L

Recall from Theorem 1 that L_1, L_2 contain random vectors from \mathbb{F}_2^n . Thus, for any pair $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$ we have $\Pr[\Delta(\mathbf{x}_1, \mathbf{x}_2) \leq \gamma n] = \binom{n}{\gamma n} \cdot 2^{-n}$. As a consequence, using a union bound over all pairs we can upper bound the size of the output list L for any NN algorithm by $|L| \leq \binom{n}{\gamma n} \cdot 2^{(2\lambda-1)n}$.

This in turn shows that the running time of Algorithm 4 is upper bounded by

$$\max \left\{ \binom{n}{\frac{\gamma}{2}n} \cdot 2^{\lambda n}, \binom{n}{\gamma n} \cdot 2^{(2\lambda-1)n} \right\}. \quad (6)$$

Since our new decoding algorithm improves the decoding with high error rate, it is best suited for attacking instances of the Learning Parity with Noise Problem (LPN).

Definition 3 (LPN). Let $\tau \in [0, \frac{1}{2})$ be some error parameter, and let $\mathbf{s} \in \mathbb{F}_2^k$ be a secret vector. In the $\text{LPN}_{k,\tau}$ problem one has oracle access to samples of the form

$$(\mathbf{a}_i, b_i) := (\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i), \text{ for } i = 1, 2, \dots$$

where $\mathbf{a}_i \in_R \mathbb{F}_2^k$ and $e_i \in \{0, 1\}$ with $\Pr[e_i = 1] = \tau$. The goal is to recover \mathbf{s} .

Let us denote by n the number of samples, which can be freely chosen. We write an LPN instance as a matrix-vector tuple

$$(\mathbf{A}, \mathbf{b}) \in \mathbb{F}_2^{n \times k} \times \mathbb{F}_2^n \text{ satisfying } \mathbf{A}\mathbf{s} = \mathbf{b} + \mathbf{e},$$

where $\mathbf{e} = (e_1, \dots, e_n)$ and the i^{th} row of A and \mathbf{b} represent the i^{th} LPN sample.

Notice that A is by definition of LPN the generator matrix of a random binary linear $[n, k]$ -code, in which we are free to choose n ourselves. Thus, we can make the rate $\frac{k}{n}$ arbitrarily small.

Moreover, \mathbf{b} is a noisy codeword that is decoded to $\mathbf{b} + \mathbf{e}$ with an error $\mathbf{e} \in \mathbb{F}_2^n$ of (large) expected weight $\mathbb{E}[\Delta(\mathbf{e})] = \tau n$. Typical parameters for τ in the cryptographic setting are $\tau = \frac{1}{4}$, or $\tau = \frac{1}{8}$.

3 The Depth-2 Algorithm

Our Goal. As described in Sect. 2, many ISD algorithms like Dumer or BJMM do an exact matching using Eq. (3) $\bar{P}_1 \mathbf{e}_1^{(1)} = \bar{P}_1 \mathbf{e}_2^{(1)} + \mathbf{s}_1$ on ℓ coordinates, and among the candidates $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}) \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{k+\ell}$ that fulfill Eq. (3), they search for those, whose remaining $n-k-\ell$ coordinates approximately match by Eq. (4).

As opposed to the BJMM algorithm, we really go back to the initial Eq. (2) $\bar{P}\mathbf{e}' + \mathbf{e}'' = \bar{\mathbf{s}}$. Splitting $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ for $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)} \in \mathbb{F}_2^k$ yields

$$\bar{P}\mathbf{e}_1^{(1)} = \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}} \text{ on all } n-k \text{ but } \Delta(\mathbf{e}'') = \omega - p \text{ coordinates.} \quad (7)$$

Our goal is to directly construct $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}$ such that $\Delta(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}) = p$ and the corresponding vectors $\bar{P}\mathbf{e}_1^{(1)}, \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}}$ *approximately* match on all $n-k$ but $\omega-p$ coordinates. This immediately yields a solution $(\mathbf{e}', \mathbf{e}'')$ with $\mathbf{e}'' = \bar{P}\mathbf{e}' + \bar{\mathbf{s}}$ and $\Delta(\mathbf{e}'') = \omega - p$ for the Decoding problem in standard form.

In comparison to other ISD algorithms, our vectors $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}$ have length k (like in Prange) instead of $k+\ell$ (like in Dumer, BJMM). This decreases the search space significantly. Moreover, it introduces a less restrictive weight distribution on a solution $(\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$, since usually $p \ll \omega$ and we only need small weight p on the first k coordinates instead of the first $k+\ell$ coordinates. This in turn means that we need less iterations in Algorithm 1 to find a permutation π that fulfills our weight distribution.

On the downside, our approximate matching routine is more costly than the exact matching in other ISD algorithms. But as our analysis shows, the benefits outweigh this disadvantage, especially when the weight of our solution is large enough.

Recall that by Eq. (7) our goal is to construct two lists $L_1^{(1)}, L_2^{(1)}$ in depth 1 of a search tree containing entries

$$\begin{aligned} & (\mathbf{e}_1^{(1)}, \bar{P}\mathbf{e}_1^{(1)}) \text{ and } (\mathbf{e}_2^{(1)}, \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}}) \text{ such that} \\ & \Delta(\mathbf{e}') = \Delta(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}) = p \text{ and } \Delta(\mathbf{e}'') = \omega - p. \end{aligned}$$

The two lists $L_1^{(1)}, L_2^{(1)}$ are constructed in a recursive manner out of other lists in a search tree of some depth m that has to be optimized. In this section, we describe our algorithm for depth $m = 2$ only, since this already gives the main ideas.

Let us introduce some useful notion, see also Fig. 2. For any vector $\mathbf{v} = v_1 \dots v_n \in \mathbb{F}_2^n$ and any positive lengths $\ell_1, \dots, \ell_{m+1} \in \mathbb{N}$ with $\sum_{j=1}^{m+1} \ell_j = n$, we define by $\mathbf{v}_{[j]} \in \mathbb{F}_2^{\ell_j}$ the *projection* of \mathbf{v} onto its coordinates $(\sum_{i=1}^{j-1} \ell_i + 1, \dots, \sum_{i=1}^j \ell_i)$. We also extend our notion to lists of vectors $L \subset \mathbb{F}_2^n$. In $L_{[j]}$ we project all elements $\mathbf{v} \in L$ to $\mathbf{v}_{[j]}$.

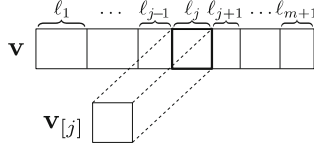


Fig. 2. The projection $\mathbf{v}_{[j]}$ of \mathbf{v} .

Outline of depth-2 algorithm. Here we give a high-level overview of our construction with a search tree of depth 2. The reader is advised to follow the description via Fig. 3.

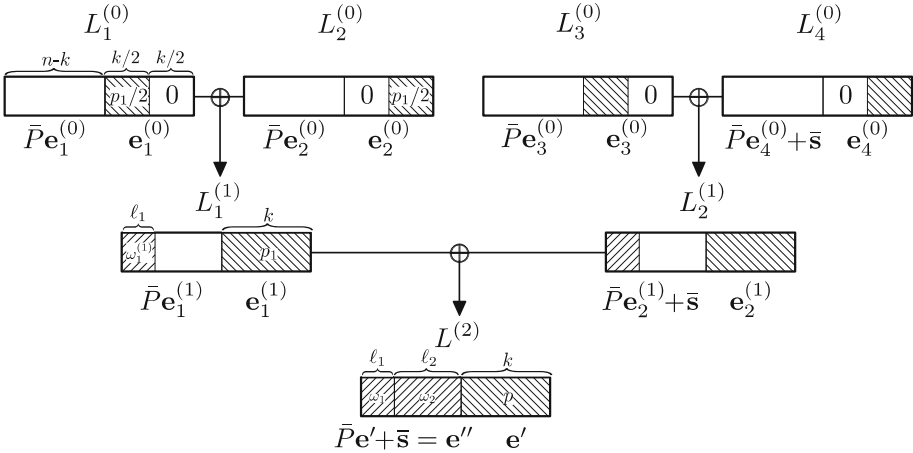


Fig. 3. Our depth-2 algorithm.

Among the $n - k$ coordinates of \mathbf{e}'' , we introduce another split into ℓ_1 and $\ell_2 := n - k - \ell_1$ coordinates. In the final list $L^{(2)}$, we enforce some weight ω_1 on the first ℓ_1 coordinates of $\mathbf{e}'' = (\mathbf{e}''_{[1]}, \mathbf{e}''_{[2]})$, and the remaining weight $\omega_2 := \omega - p - \omega_1$ on the remaining ℓ_2 coordinates. The parameters ℓ_1, ω_1 are subject to optimization.

For the construction of $\mathbf{e}''_{[2]}$ we use an NN search for the lists $L_1^{(1)}, L_2^{(1)}$ on level 1 that gives us weight ω_2 on these coordinates.

In those lists $L_1^{(1)}, L_2^{(1)}$ we furthermore enforce weight $p_1 \geq \frac{p}{2}$ on the coordinates of $\mathbf{e}_1^{(1)}$ and $\mathbf{e}_2^{(1)}$. The parameter p_1 is again subject to optimization.

Analogously we restrict to only those $\bar{P}\mathbf{e}_1^{(1)}, \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}}$ whose first ℓ_1 coordinates have a weight $\omega_1^{(1)}$. The weight $\omega_1^{(1)}$ has to be optimized. Again, we filter out all vector sums on level 2 whose weight is not exactly ω_1 on these ℓ_1 coordinates.

The lists $L_1^{(1)}, L_2^{(1)}$ are constructed out of four lists $L_i^{(0)}, i = 1, \dots, 4$ on level 0. Here, we describe only the construction of $L_1^{(1)}$, the construction of $L_2^{(1)}$ is analogous. In $L_1^{(0)}$ we enumerate all vectors $\mathbf{e}_1^{(0)} \in \mathbb{F}_2^{k/2} \times 0^{k/2}$ with weight $p_1/2$. For each of these vectors we compute $\bar{P}\mathbf{e}_1^{(0)}$. Similarly, in $L_2^{(0)}$ we enumerate all vectors $\mathbf{e}_2^{(0)} \in 0^{k/2} \times \mathbb{F}_2^{k/2}$ with weight $p_1/2$ and compute $\bar{P}\mathbf{e}_2^{(0)}$. We then run a NN search on the first ℓ_1 coordinates to find all vector sums with weight $\omega_1^{(1)}$ on these coordinates. Note that the vectors $\mathbf{e}_1^{(0)}$ and $\mathbf{e}_2^{(0)}$ automatically add up to a vector of weight p_1 as required for list $L_1^{(1)}$.

This concludes the high-level description of our algorithm. More details can be found in Algorithm 5, which has to be used as an ISDSOLVE-subroutine in Algorithm 1 to obtain a full fletched ISD algorithm, including column permutation π and transformation to standard form.

List of objects. For completeness, we provide in the following a precise description of the lists. For the lists of level 0, we have

$$\begin{aligned} L_1^{(0)} &= \{(\bar{P}\mathbf{e}_1^{(0)}, \mathbf{e}_1^{(0)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^{k/2} \times 0^{k/2} \mid \Delta(\mathbf{e}_1^{(0)}) = p_1/2\}, \\ L_2^{(0)} &= \{(\bar{P}\mathbf{e}_2^{(0)}, (\mathbf{e}_2^{(0)})) \in \mathbb{F}_2^{n-k} \times 0^{k/2} \times \mathbb{F}_2^{k/2} \mid \Delta(\mathbf{e}_2^{(0)}) = p_1/2\}, \\ L_3^{(0)} &= \{(\bar{P}\mathbf{e}_3^{(0)}, \mathbf{e}_3^{(0)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^{k/2} \times 0^{k/2} \mid \Delta(\mathbf{e}_3^{(0)}) = p_1/2\}, \\ L_4^{(0)} &= \{(\bar{P}\mathbf{e}_4^{(0)} + \bar{\mathbf{s}}, \mathbf{e}_4^{(0)}) \in \mathbb{F}_2^{n-k} \times 0^{k/2} \times \mathbb{F}_2^{k/2} \mid \Delta(\mathbf{e}_4^{(0)}) = p_1/2\}. \end{aligned} \quad (8)$$

Thus, all lists on level 0 have size $S_0 = \binom{k/2}{p_1/2}$. Note that $L_1^{(0)} = L_3^{(0)}$. The lists on level 1 are constructed via NN search on the first ℓ_1 coordinates such that we obtain weight $\omega_1^{(1)}$ on these coordinates. This yields

$$\begin{aligned} L_1^{(1)} &= \{(\bar{P}\mathbf{e}_1^{(1)}, \mathbf{e}_1^{(1)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^k \mid \Delta(\mathbf{e}_1^{(1)}) = p_1, \Delta((\bar{P}\mathbf{e}_1^{(1)})_{[1]}) = \omega_1^{(1)}\}, \\ L_2^{(1)} &= \{(\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}}, \mathbf{e}_2^{(1)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^k \mid \Delta(\mathbf{e}_2^{(1)}) = p_1, \Delta((\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]}) = \omega_1^{(1)}\}. \end{aligned}$$

By the randomness of \bar{P} , both lists have expected size

$$\begin{aligned} S_1 &:= \mathbb{E}[|L_i^{(1)}|] = S_0^2 \cdot \Pr[\text{weight } \omega_1^{(1)} \text{ on the first } \ell_1 \text{ coordinates}] \\ &= \binom{k/2}{p_1/2}^2 \cdot \frac{\binom{\ell_1}{\omega_1^{(1)}}}{2^{\ell_1}} \quad \text{for } i = 1, 2. \end{aligned}$$

Eventually, by an NN search on ℓ_2 bits for weight ω_2 on the level-1 lists and subsequent filtering for weight p on the last k coordinates and weight ω_1 on the first ℓ_1 bits, we obtain

$$L^{(2)} = \{(\mathbf{e}'', \mathbf{e}') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k} \mid \Delta(\mathbf{e}') = p, \mathbf{e}'' = \bar{P}\mathbf{e}' + \bar{\mathbf{s}}, \Delta(\mathbf{e}'') = \omega - p\}.$$

Thus, any element $(\mathbf{e}'', \mathbf{e}')$ of $L^{(2)}$ yields a solution $(\mathbf{e}', \mathbf{e}'')$ of a Syndrome Decoding Problem in standard form.

Algorithm 5. DEPTH-2-ISDSOLVE

Input : $\bar{P} \in \mathbb{F}_2^{(n-k) \times k}, \bar{\mathbf{s}} \in \mathbb{F}_2^{n-k}, \omega$
Output : $(\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$
Parameters: Optimize $p, \omega_1, \ell_1, p_1, \omega_1^{(1)}$.
 Set $\omega_2 = \omega - p - \omega_1$ and $\ell_2 = n - k - \ell_1$.

- 1 Create lists $L_i^{(0)}, i = 1, 2, 3, 4$ as defined in (8)
- 2 $L_i^{(1)} \leftarrow \text{NN-Search}(L_{2i-1}^{(0)}, L_{2i}^{(0)}, 1, \omega_1^{(1)}), i = 1, 2$
 $\triangleright \text{NN-Search}(L_1, L_2, i, w)$ performs a NN search on $(L_1)_{[i]}, (L_2)_{[i]}$
 \triangleright with target weight w while keeping all other coordinates.
- 3 $L^{(2)} \leftarrow \text{NN-Search}(L_1^{(1)}, L_2^{(1)}, 2, \omega_2)$
- 4 $L^{(2)} \leftarrow \text{Filter}(L^{(2)}, 1, \omega_1)$ $\triangleright \text{Filter}(L, i, w)$ filters L for elements
 $L^{(2)} \leftarrow \text{Filter}(L^{(2)}, 3, p)$ \triangleright with weight w on its projection in $L_{[i]}$.

if $|L^{(2)}| > 0$ **then return** $(\mathbf{e}', \mathbf{e}'')$ for some $(\mathbf{e}'', \mathbf{e}') \in L^{(2)}$
else return \perp

Notice that Algorithm 5 can only succeed to output a solution $(\mathbf{e}', \mathbf{e}'') \neq \perp$ if there exists some \mathbf{e}' with weight p such that $\bar{P}\mathbf{e}' + \bar{\mathbf{s}} = \mathbf{e}'' = (\mathbf{e}''_{[1]}, \mathbf{e}''_{[2]})$ with $\mathbf{e}''_{[1]}, \mathbf{e}''_{[2]}$ having weights ω_1 and ω_2 , respectively. This specific weight distribution has to be induced by the column permutation π of Algorithm 1.

Definition 4. Let $\mathbf{e} \in \mathbb{F}_2^n$ with $\Delta(\mathbf{e}) = \omega$ and $k, p \in \mathbb{N}$. Let $\ell_1, \ell_2 \in \mathbb{N}$ with $\ell_1 + \ell_2 = n - k$, and let $\omega_1, \omega_2 \in \mathbb{N}$ with $\omega_1 + \omega_2 = \omega - p$. We call a permutation π good for \mathbf{e} with respect to $(p, \omega_1, \ell_1, \omega_2, \ell_2)$, if $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}''_{[1]}, \mathbf{e}''_{[2]}) \in \mathbb{F}_2^k \times \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell_2}$ with

$$\Delta(\mathbf{e}') = p, \quad \Delta(\mathbf{e}''_{[1]}) = \omega_1 \text{ and } \Delta(\mathbf{e}''_{[2]}) = \omega_2.$$

A random permutation π is good with probability

$$P_\pi = \frac{\binom{k}{p} \binom{\ell_1}{\omega_1} \binom{\ell_2}{\omega_2}}{\binom{n}{\omega}}.$$

It remains to show that on input a standard form Syndrome Decoding instance $(\bar{P}, \bar{\mathbf{s}}, \omega)$ that stems from a good π , Algorithm 5 constructs a non-empty list of solutions $L^{(2)}$.

Lemma 1 (Correctness). *Let \mathbf{e} be a solution to the Syndrome Decoding Problem. Let π be good for \mathbf{e} with respect to any fixed parameters $(p, \omega_1, \ell_1, \omega_2, \ell_2)$ as given by Definition 4. Whenever we run Algorithm 5 with parameters $p_1, \omega_1^{(1)} \in \mathbb{N}$ satisfying*

$$\binom{p}{p/2} \binom{k-p}{p_1-p/2} \geq \frac{2^{\ell_1}}{\binom{\omega_1}{\omega_1/2} \binom{\ell_1-\omega_1}{\omega_1^{(1)}-\omega_1/2}}, \quad (9)$$

then on expectation we have $(\mathbf{e}'', \mathbf{e}') \in L^{(2)}$ for $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}'')$.

Thus, Lemma 1 shows that any (possibly unique) solution \mathbf{e} to the Syndrome Decoding Problem is constructed in our sub-routine of Algorithm 5 at that point in time when the full-fledged ISD Algorithm 1 provides a good permutation π , under the condition that (9) holds.

Before we prove Lemma 1, we would like to show that its statement is not vacuous. Namely, there always exist $p_1, \omega_1^{(1)}$ such that condition (9) holds.

Using the Binomial Theorem, we have

$$\binom{n}{n/2} < \sum_{i=0}^n \binom{n}{i} = 2^n < (n+1) \binom{n}{n/2}.$$

This implies

$$\frac{2^n}{n+1} < \binom{n}{n/2} < 2^n.$$

Thus, up to a linear factor we can approximate $\binom{n}{n/2}$ by 2^n . Hence if we ignore linear factors, condition (9) collapses for the setting $p_1 = k/2$ and $\omega_1^{(1)} = \ell_1/2$ to

$$2^{p+k-p} \geq 2^{\ell_1-\omega_1-(\ell_1-\omega_1)} \Leftrightarrow k \geq 0,$$

which is trivially fulfilled. Thus, there always exist feasible parameters $p, \omega_1, \ell_1, p_1, \omega_1^{(1)}$ that lead to a solution when running Algorithm 5. Among these feasible parameters, we will later minimize running time.

Proof (of Lemma 1). Let $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}'')$ be the solution of our Syndrome Decoding problem in standard form. Since we have standard form, we conclude that $\mathbf{e}'' = \bar{P}\mathbf{e}' + \bar{\mathbf{s}}$ is fully determined by \mathbf{e}' . Moreover, since we assume π to be good, \mathbf{e}'' is of the correct form. Thus, it suffices to show that Algorithm 5 constructs the desired $\mathbf{e}' \in \mathbb{F}_2^k$.

Notice that in our construction $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$, and in turn $\mathbf{e}_1^{(1)} = \mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)}$ (and analogous for $\mathbf{e}_2^{(1)}$).

Let us first argue that in our construction we obtain up to a polynomial factor all $\binom{k}{p_1}$ vectors $\mathbf{e}_1^{(1)} \in \mathbb{F}_2^k$ on level 1. All vector sums $\mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)}$ are by the definition of $\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)}$ different. Up to polynomial factors (denoted by \approx), we have by standard approximation via the binary entropy function $\binom{k/2}{p_1/2}^2 \approx 2^{2H(\frac{p_1}{k})k/2} \approx \binom{k}{p_1}$ vectors $\mathbf{e}_1^{(1)}$ that we construct.

Now let us turn to the construction of \mathbf{e}' with weight p on level 2 via $\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ with $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}$ having weight $p_1 \geq p/2$. We call $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ a representation of \mathbf{e}' if $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$. Notice that our desired solution \mathbf{e}' has

$$R_2 := \binom{p}{p/2} \binom{k-p}{p_1-p/2} \text{ representations,} \quad (10)$$

since the set of 1-coordinates in \mathbf{e}' can be represented in $\binom{p}{p/2}$ ways as $1+0$ or $0+1$, and the set of 0-coordinates in \mathbf{e}' can be represented in $\binom{k-p}{p_1-p/2}$ ways as $0+0$ or $1+1$.

From an algorithmic point of view, we do not care which of the R_2 representations is eventually used for constructing \mathbf{e}' . It is therefore sufficient that only 1 of these R_2 representations is present in $L_1^{(1)} \times L_2^{(1)}$. Hence, for achieving minimal run time we construct only a random $1/R_2$ -fraction of all representations such that on expectation one representation is present in $L_1^{(1)} \times L_2^{(1)}$, and therefore \mathbf{e}' appears in $L^{(2)}$.

For constructing only an $1/R_2$ -fraction, we construct on level 1 only those elements $(\bar{P}\mathbf{e}_1^{(1)}, \mathbf{e}_1^{(1)}) \in L_1^{(1)}$ whose first ℓ_1 coordinates have weight $\omega_1^{(1)}$ (analogous for $L_2^{(1)}$). This means that we enforce $\Delta((\bar{P}\mathbf{e}_1^{(1)})_{[1]}) = \omega_1^{(1)}$. Let E be the event that there exists a representation of

$$\mathbf{e}''_{[1]} = (\bar{P}\mathbf{e}_1^{(1)} + \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]} \text{ with } \Delta((\bar{P}\mathbf{e}_1^{(1)})_{[1]}) = \Delta((\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]}) = \omega_1^{(1)}.$$

By randomness of \bar{P} , we have

$$p_{2,2} := \Pr[E] = \frac{\binom{\omega_1}{\omega_1/2} \binom{\ell_1 - \omega_1}{\omega_1^{(1)} - \omega_1/2}}{2^{\ell_1}},$$

since there are a total of 2^{ℓ_1} possible representations of the form $\mathbf{e}_{[1]} = (\bar{P}\mathbf{e}_1^{(1)} + \bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]}$ out of which $\binom{\omega_1}{\omega_1/2} \binom{\ell_1 - \omega_1}{\omega_1^{(1)} - \omega_1/2}$ have the correct weight $\omega_1^{(1)}$ for $(\bar{P}\mathbf{e}_1^{(1)})_{[1]}, (\bar{P}\mathbf{e}_2^{(1)} + \bar{\mathbf{s}})_{[1]}$ by the same argument as in Eq. (10).

Thus, the expected number of representations of \mathbf{e}' is $R_2 \cdot p_{2,2}$. Hence on expectation, we construct \mathbf{e}' in $L^{(2)}$ if $R_2 \cdot p_{2,2} \geq 1$, which is equivalent to condition (9). \square

Complexity of the Depth-2 Algorithm. Our Algorithm 5 starts with the construction of lists $L_i^{(0)}, i = 1, 2, 3, 4$ (step 1) which takes time $S_0 = \binom{k/2}{p_1/2}$. By Theorem 1 and Eqs. (5), (6), the Nearest Neighbor search on $(L_i^{(0)})_{[1]}$ (step 2) takes time

$$T_0 := \begin{cases} 2y^{\left(\frac{\log(S_0)}{\ell_1}, \frac{\omega_1^{(1)}}{\ell_1}\right)\ell_1} & \text{if } \frac{\log(S_0)}{\ell_1} < 1 - H\left(\frac{\omega_1^{(1)}}{2\ell_1}\right) \\ \min\{S_0^2, \max\left\{\left(\frac{\ell_1}{2}\right) \cdot S_0, S_0^2 \cdot \left(\frac{\omega_1^{(1)}}{2\ell_1}\right)\right\}\} & \text{else} \end{cases}.$$

resulting in the lists $L_1^{(1)}, L_2^{(1)}$ that have expected size $S_1 = \binom{k/2}{p_1/2}^2 \cdot \frac{\binom{\ell_1}{\omega_1^{(1)}}}{2^{\ell_1}}$. These lists are again combined via Nearest Neighbor search (step 3) in time

$$T_1 := \begin{cases} 2^{y(\frac{\log(S_1)}{\ell_2}, \frac{\omega_2^2}{\ell_2})\ell_2} & \text{if } \frac{\log(S_1)}{\ell_2} < 1 - H(\frac{\omega_2}{2\ell_2}) \\ \min\{S_1^2, \max\{(\frac{\ell_2}{2}) \cdot S_1, S_1^2 \cdot \frac{\binom{\ell_2}{\omega_2}}{2^{\ell_2}}\}\} & \text{else} \end{cases},$$

resulting in the final list $L^{(2)}$. The filtering (step 4) takes time $S_2 := |L^{(2)}|$. We only need to store the lists $L_i^{(0)}$ of size S_0 as well as the lists $L_1^{(1)}, L_2^{(1)}$ of size S_1 . The total running time is

$$\text{time } T = \max\{T_0, T_1, S_0, S_2\} = \max\{T_0, T_1\},$$

since $T_0 \geq S_0$ and $T_1 \geq S_2$.

Total complexity of Decoding. Algorithm 5 constructs a solution iff π is good which happens according to Definition 4 with probability P_π , resulting in a total expected running time of $T \cdot P_\pi^{-1}$ for our full-fledged ISD algorithm.

4 The Depth- m Algorithm

Our algorithm with depth 2, as described in the previous Sect. 3, already illustrates the overall idea of approximate matching, but does not yet yield improved running times compared to the BJMM algorithm. Therefore, we generalize to arbitrary depth in this section, which is mostly straight-forward but still includes some subtleties how to proceed with approximate matchings - and their respective weights - over many levels of a search tree.

Outline of depth- m algorithm. Let us start again with a high-level overview for our algorithm with arbitrary depth m . The reader is advised to follow the description via Fig. 4 which shows the algorithm for $m = 3$.

In the final list $L_1^{(m)}$ we now split the first $n - k$ coordinates into m blocks instead of only 2 blocks, i.e we have $\mathbf{e}'' = (\mathbf{e}''_{[1]}, \dots, \mathbf{e}''_{[m]})$. Block $\mathbf{e}''_{[i]}$ has length ℓ_i and weight $\omega_i^{(m)}$. The parameters $\ell_i, \omega_i^{(m)}$ are subject to optimization.

On level 0, there are a total of 2^m lists $L_1^{(0)}, \dots, L_{2^m}^{(0)}$. The construction of the 2^{m-1} lists $L_1^{(1)}, \dots, L_{2^{m-1}}^{(1)}$ on level 1 out of the level-0 lists is identical to the construction in Sect. 3.

For the level- m lists, we have

$$\begin{aligned} L_{j_1}^{(0)} &= \{(\bar{P}\mathbf{e}_{j_1}^{(0)}, \mathbf{e}_{j_1}^{(0)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^{k/2} \times 0^{k/2} \mid \Delta(\mathbf{e}_{j_1}^{(0)}) = p_1/2\}, & (11) \\ L_{j_2}^{(0)} &= \{(\bar{P}\mathbf{e}_{j_2}^{(0)}, \mathbf{e}_{j_2}^{(0)}) \in \mathbb{F}_2^{n-k} \times 0^{k/2} \times \mathbb{F}_2^{k/2} \mid \Delta(\mathbf{e}_{j_2}^{(0)}) = p_1/2\}, \\ L_{2^m}^{(0)} &= \{(\bar{P}\mathbf{e}_{2^m}^{(0)} + \bar{\mathbf{s}}, \mathbf{e}_{2^m}^{(0)}) \in \mathbb{F}_2^{n-k} \times 0^{k/2} \times \mathbb{F}_2^{k/2} \mid \Delta(\mathbf{e}_{2^m}^{(0)}) = p_1/2\}. \end{aligned}$$

for $j_1 = 1, 3, \dots, 2^m - 1$ and $j_2 = 2, 4, \dots, 2^m - 2$. All lists on level 0 therefore have size $S_0 = \binom{k/2}{p_1/2}$.

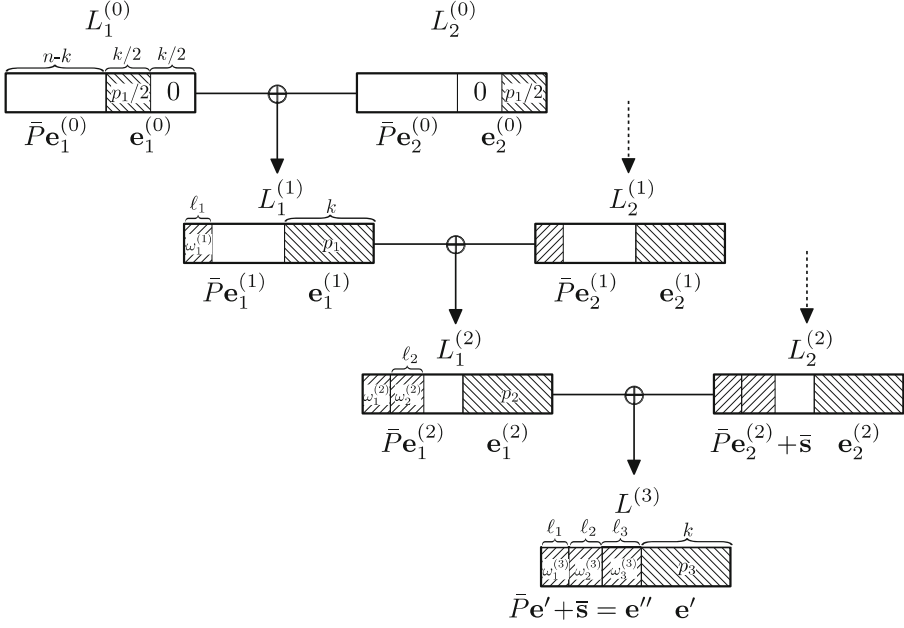


Fig. 4. Our algorithm for depth 3.

Starting from the level-0 lists, our algorithm combines two lists at a time using NN search in a binary tree wise fashion until we reach the final list $L^{(m)}$. On every level $i = 1, \dots, m - 1$ we construct the first list $L_1^{(i)}$ via NN search on the projected lists $(L_1^{(i-1)})_{[i]}, (L_2^{(i-1)})_{[i]}$ such that we obtain weight $\omega_i^{(i)}$. We furthermore filter for weight p_i on the last k coordinates and a specific weight distribution on the remaining coordinates such that we get

$$L_1^{(i)} = \{(\bar{P}\mathbf{e}_1^{(i)}, \mathbf{e}_1^{(i)}) \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^k \mid \Delta(\mathbf{e}_1^{(i)}) = p_i, \Delta((\bar{P}\mathbf{e}_1^{(i)})_{[h]}) = \omega_h^{(i)}, h = 1, \dots, i\}.$$

The other lists $L_j^{(i)}, j = 2, \dots, 2^i$ are created analogously. By randomness of \bar{P} the projection $(\bar{P}\mathbf{e}_1^{(i)})_{[i]}$ with weight $\omega_i^{(i)}$ is in our construction the sum of two random vectors. For every $h = 1, \dots, i - 1$ the projection $(\bar{P}\mathbf{e}_1^{(i)})_{[h]}$ with weight $\omega_h^{(i)}$ is the sum of two random vectors of specific weight $\omega_h^{(i-1)}$. Fixing the first vector, there are $\binom{\ell_h}{\omega_h^{(i-1)}}$ possible second vectors out of which $\binom{\omega_h^{(i-1)}}{\omega_h^{(i-1)} - \omega_h^{(i)}/2} \binom{\ell_h - \omega_h^{(i-1)}}{\omega_h^{(i)}/2}$ yield the correct weight $\omega_h^{(i)}$. Therefore, the expected list sizes on layer i are upper bounded by

$$\begin{aligned}
S_i &\leq |\{\mathbf{x} \in \mathbb{F}_2^k \mid \Delta(\mathbf{x}) = p_i\}| \cdot \Pr_{\mathbf{x} \in \mathbb{F}_2^{\ell_i}} [\Delta(\mathbf{x}) = \omega_i^{(i)}] \\
&\cdot \prod_{h=i+1}^{m-1} \Pr_{\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^{\ell_h}} [\Delta(\mathbf{x} + \mathbf{y}) = \omega_h^{(i)} \mid \Delta(\mathbf{x}) = \Delta(\mathbf{y}) = \omega_h^{(i+1)}] \\
&= \binom{k}{p_i} \cdot \frac{\binom{\ell_i}{\omega_i^{(i)}}}{2^{\ell_i}} \cdot \prod_{h=1}^{i-1} \frac{\binom{\ell_h - \omega_h^{(i-1)}}{\omega_h^{(i-1)} - \omega_h^{(i)}/2} \binom{\ell_h - \omega_h^{(i-1)}}{\omega_h^{(i)}/2}}{\binom{\ell_h}{\omega_h^{(i-1)}}}.
\end{aligned}$$

Eventually, an NN search on $\ell_m = n - k - \sum_{i=1}^{m-1} \ell_i$ bits for weight $\omega_m^{(m)} = \omega - p_m - \sum_{i=1}^{m-1} \omega_i^{(m)}$, $p_m = p$, on the level- $(m-1)$ lists and subsequent filtering for weight p_m on the last k coordinates and weight $\omega_i^{(m)}$ for every projection $\mathbf{e}''_{[i]}$, $i = 1, \dots, m-1$, we obtain

$$L^{(m)} = \{(\mathbf{e}'', \mathbf{e}') \in \mathbb{F}_2^{n-k} \times \mathbb{F}_2^k \mid \Delta(\mathbf{e}') = p_m, \mathbf{e}'' = \bar{P}\mathbf{e}' + \bar{\mathbf{s}}, \Delta(\mathbf{e}'') = \omega - p_m\}.$$

Thus, any element $(\mathbf{e}'', \mathbf{e}')$ of $L^{(m)}$ yields a solution $(\mathbf{e}', \mathbf{e}'')$ of a Syndrome Decoding Problem in standard form.

More details can be found in Algorithm 6, which has to be used again as an ISDSOLVE-subroutine in Algorithm 1 to obtain a full fetched ISD algorithm.

Algorithm 6. DEPTH- m -ISDSOLVE

Input : $\bar{P} \in \mathbb{F}_2^{(n-k) \times k}$, $\bar{\mathbf{s}} \in \mathbb{F}_2^{n-k}$, ω

Output : $(\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$

Parameters: Optimize $p_1, \dots, p_m, \omega_1^{(m)}, \dots, \omega_{m-1}^{(m)}, \ell_1, \dots, \ell_{m-1}$.

Compute $\omega_m^{(m)} = \omega - p_m - \sum_{i=1}^{m-1} \omega_i^{(m)}$, $\ell_m = n - k - \sum_{i=1}^{m-1} \ell_i$.

- 1 Define $\omega_i^{(i)} := \frac{\omega_i^{(i+1)}}{2}$, $i = 1, \dots, m-2$.
Choose optimal $\omega_j^{(i)}$ such that condition (12) holds.
 - 2 Create lists $L_j^{(0)}$, $j = 1, \dots, 2^m$ as defined in (11).
 - 3 $L_j^{(1)} \leftarrow \text{NN-Search}((L_{2j-1}^{(0)})_{[1]}, (L_{2j}^{(0)})_{[1]}, \omega_1^{(1)})$, $j = 1, \dots, 2^{m-1}$
for $i = 2, \dots, m$, $j = 1, \dots, 2^i$ **do**
 - 4 $L_j^{(i)} \leftarrow \text{NN-Search}((L_{2j-1}^{(i-1)})_{[i]}, (L_{2j}^{(i-1)})_{[i]}, \omega_i^{(i)})$
 - 5 $L_j^{(i)} \leftarrow \text{Filter}(L_j^{(i)}, h, \omega_h^{(i)})$, $h = 1, \dots, i-1$
 $L_j^{(i)} \leftarrow \text{Filter}(L_j^{(i)}, m+1, p_i)$
 - end**
 - if** $|L^{(m)}| > 0$ **then return** $(\mathbf{e}', \mathbf{e}'')$ for some $(\mathbf{e}'', \mathbf{e}') \in L^{(m)}$
 - else return** \perp
-

Notice that Algorithm 6 can only succeed to output a solution $(\mathbf{e}', \mathbf{e}'') \neq \perp$ if there exists some \mathbf{e}' with weight p_m such that $\bar{P}\mathbf{e}' + \bar{\mathbf{s}} = (\mathbf{e}''_{[1]}, \dots, \mathbf{e}''_{[m]})$ with $\mathbf{e}''_{[i]}$ having weight $\omega_i^{(m)}$ for all $i = 1, \dots, m$. This specific weight distribution has to be induced by the column permutation π of Algorithm 1.

Definition 5. Let $\mathbf{e} \in \mathbb{F}_2^n$ with $\Delta(\mathbf{e}) = \omega$ and $k, p_m \in \mathbb{N}$. Let $\ell_1, \dots, \ell_m \in \mathbb{N}$ with $\sum_{i=1}^m \ell_i = n - k$, and $\omega_1^{(m)}, \dots, \omega_m^{(m)} \in \mathbb{N}$ with $\sum_{i=1}^m \omega_i^{(m)} = \omega - p_m$.

We call a permutation π good for \mathbf{e} with respect to $p_m, (\omega_i^{(m)}, \ell_i)_{i=1, \dots, m}$, if $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}''_{[1]}, \dots, \mathbf{e}''_{[m]}) \in \mathbb{F}_2^k \times \mathbb{F}_2^{\ell_1} \times \dots \times \mathbb{F}_2^{\ell_m}$ with

$$\Delta(\mathbf{e}') = p_m, \quad \Delta(\mathbf{e}''_{[i]}) = \omega_i^{(m)}, i = 1, \dots, m.$$

A random permutation π is good with probability

$$P_\pi = \frac{\binom{k}{p_m} \prod_{i=1}^m \binom{\ell_i}{\omega_i^{(m)}}}{\binom{n}{\omega}}.$$

We now show that on input a standard form Syndrome Decoding instance $(\bar{P}, \bar{\mathbf{s}}, \omega)$ that stems from a good π , Algorithm 6 constructs a non-empty list of solutions $L^{(m)}$.

Lemma 2 (Correctness). Let \mathbf{e} be a solution to the Syndrome Decoding Problem. Let π be good for \mathbf{e} with respect to any fixed parameters $p_m, \omega_i^{(m)}, \ell_i, i = 1, \dots, m$ as given by Definition 5. Whenever we run Algorithm 6 with parameters $p_i, \omega_j^{(i)} \in \mathbb{N}$, for $j = 1, \dots, i, i = 1, \dots, m - 1$ satisfying

$$\binom{p_i}{p_i/2} \binom{k - p_i}{p_{i-1} - p_i/2} \geq \prod_{h=1}^{i-1} \frac{2^{\ell_h}}{\binom{\omega_h^{(i)}}{\omega_h^{(i)}/2} \binom{\ell_h - \omega_h^{(i)}}{\omega_h^{(i-1)} - \omega_h^{(i)}/2}}, \quad \forall i = 2, \dots, m \quad (12)$$

then on expectation we have $(\mathbf{e}'', \mathbf{e}') \in L^{(m)}$ for $\pi(\mathbf{e}) = (\mathbf{e}', \mathbf{e}'')$.

Analogous to Sect. 3, we can show that the setting $p_i = k/2$ and $\omega_h^{(i)} = \ell_h/2$, for $h = 1, \dots, i - 1, i = 2, \dots, m$, yields feasible parameters for Algorithm 6 that fulfill condition (12).

Proof (of Lemma 2). The proof is similar to the proof of Lemma 1 and can be found in the full version [BM17a].

Complexity of Algorithm 6. The lists $L_j^{(0)}, j = 1, \dots, 2^m$ are created in time S_0 (step 2). The NN search on those lists yields lists $L_j^{(1)}, j = 1, \dots, 2^{m-1}$ (step 3).

Next, another NN search on the new lists returns lists $L_j^{(2)}, j = 1, \dots, 2^{m-2}$ (step 4) which are subsequently filtered (step 5). These two steps of NN search and filtering are repeated until only one list is left. By Theorem 1 and Eqs. (5), (6) the NN search layer on $i = 0, \dots, m - 1$ takes time

$$T_i := \begin{cases} 2^{y \binom{\log(S_i)}{\ell_{i+1}}, \frac{\omega_{i+1}^{(i+1)}}{\ell_{i+1}}} \ell_{i+1} & \text{if } \frac{\log(S_i)}{\ell_{i+1}} < 1 - H\left(\frac{\omega_{i+1}^{(i+1)}}{2\ell_{i+1}}\right) \\ \min\{S_i^2, \max\{\left(\frac{\ell_{i+1}}{\omega_{i+1}^{(i+1)}}\right) \cdot S_i, S_i^2 \cdot \frac{\omega_{i+1}^{(i+1)}}{2\ell_{i+1}}\}\} & \text{else} \end{cases}.$$

The filtering takes time S_i on layer $i = 2, \dots, m-1$ and $S_m := |L^{(m)}|$ on layer m .

On every level i of our search tree we consume time T_i and store lists of size S_i . Thus, we obtain

$$\text{time } T = \max_{i=1, \dots, m} \{T_i\}$$

using $T_i \geq S_i$ for $i = 0, \dots, m-1$ and $T_{m-1} \geq S_m$.

Total complexity of Decoding. Algorithm 6 constructs a solution iff π is good which happens according to Definition 5 with probability P_π , resulting in a total expected running time of $T \cdot P_\pi^{-1}$ for our full-fledged ISD algorithm.

5 Results

Syndrome Decoding Problem. The best known complexity for Full Distance decoding is currently $2^{0.0953n}$ using BJMM in depth 4 [BM17b], whereas for Half Distance Decoding the best known bound is $2^{0.0473n}$ [MO15].

As stated in Theorem 2, we improve the bound for Full Distance Decoding to $2^{0.885n}$. In the Half Distance Decoding setting, we achieve a small improvement to $2^{0.0465n}$.

Theorem 2. *Algorithm 1 in combination with Algorithm 6 for $m = 4$ solves Full Distance decoding for random binary linear codes in expected time $2^{0.0885n}$ using $2^{0.0736n}$ space. Half Distance decoding is solved in expected time $2^{0.0465n}$ using $2^{0.0294n}$ space.*

Proof. For Full Distance Decoding we achieve the maximal running time at code rate

$$\frac{k}{n} = 0.46 \text{ with relative distance } \frac{\omega}{n} = \frac{d}{n} = H^{-1}\left(1 - \frac{k}{n}\right) = 0.1237.$$

For this code rate, we minimize the running time choosing the relative weights

$$\frac{p_1}{n} = 0.00559, \quad \frac{p_2}{n} = 0.01073, \quad \frac{p_3}{n} = 0.02029, \quad \frac{p_4}{n} = 0.03460,$$

resulting in

$$R_2 = 2^{0.01357n}, \quad R_3 = 2^{0.02668n}, \quad R_4 = 2^{0.06028n}$$

representations. Furthermore we set

$$\frac{\ell_1}{n} = 0.0366, \quad \frac{\ell_2}{n} = 0.0547, \quad \frac{\ell_3}{n} = 0.0911,$$

$$\frac{\omega_1}{n} = 0.0066, \quad \frac{\omega_2}{n} = 0.0099, \quad \frac{\omega_3}{n} = 0.0114, \quad \frac{\omega_1^{(3)}}{n} = 0.0232.$$

Optimization showed that

$$\omega_1^{(1)} = \frac{\omega_1^{(2)}}{2}, \quad \omega_2^{(2)} = \frac{\omega_2^{(3)}}{2}$$

is a good choice which yields

$$\frac{\omega_1^{(1)}}{n} = 0.011515, \quad \frac{\omega_1^{(2)}}{n} = 0.023029,$$

$$\frac{\omega_2^{(2)}}{n} = 0.016676, \quad \frac{\omega_2^{(3)}}{n} = 0.033351, \quad \frac{\omega_3^{(3)}}{n} = 0.009993$$

using condition Eq. (12) from Lemma 2. The resulting list sizes are

$$S_0 = 2^{0.02179n}, \quad S_1 = 2^{0.03987n}, \quad S_2 = 2^{0.05939n}, \quad S_3 = 2^{0.05975n}.$$

The lists on layer 0 are combined with the NN search of Algorithm 3 in time

$$T_0 = 2^{0.04359n},$$

as the condition for May-Ozerov is not satisfied and Algorithm 4 is less efficient in this case. On layer 1 we use Algorithm 4 in time

$$T_1 = 2^{0.07356n},$$

which is also the space consumption for this step. On the remaining layers, we use May-Ozerov NN search which yields

$$T_2 = 2^{0.07365n}, \quad T_3 = 2^{0.07359n}.$$

The probability for the correct weight distribution satisfying Definition 5 is

$$P_\pi = 2^{-0.01485n}.$$

Thus the overall running time and space consumption is

$$T = \frac{T_2}{P_\pi} = 2^{0.0885n} \quad \text{and} \quad S = T_1 = 2^{0.0736n}.$$

The case of Half Distance Decoding is analogous and can be found in the full version [BM17a]. \square

While Theorem 2 states the run time for the worst-case rate, Fig. 5 illustrates and compares the run time as a function of all constant rates $\frac{k}{n}$ of our algorithm to other decoding algorithms like Prange, BJMM and BJMM with NN-search, called BJMM-NN.

We also provide the C-code for optimizing all these algorithms at <https://github.com/LeifBoth/Decoding-LPN>.

Figure 6 compares in more detail for varying depths m the complexity of our algorithm to BJMM-NN, as analyzed in [BM17b]. Here, we consider FD, HD and typical McEliece instances with $k = 0.775$ and $\omega = 0.02$ [BLP08].

In the Full Distance (FD) setting, our algorithm is superior to BJMM-NN in all depths $m = 2, 3, 4$. Already for depth $m = 3$, we beat the current FD record. Moreover, the improvement of the exponent from $0.0953n$ to only $0.0885n$ is quite

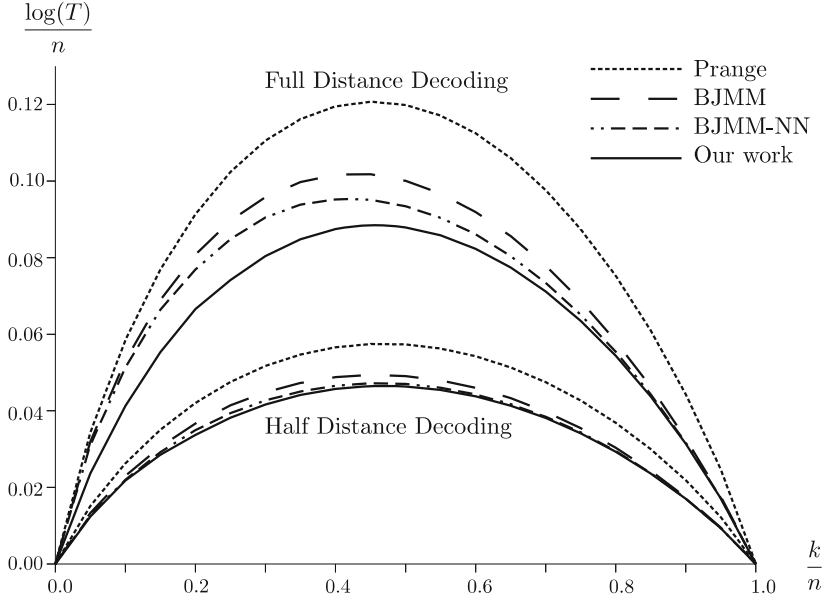


Fig. 5. [Pra62, BJMM12, MO15] and our algorithm for varying code rates $\frac{k}{n}$.

significant. Another quite surprising benefit of our algorithm when compared to BJMM-NN is its modest space consumption. We were not able to improve our running times for $m = 5$, due to the large parameter space for optimization. Whether further improvements are possible currently remains an open problem.

In the Half Distance (HD) setting, our algorithm also outperforms BJMM-NN, slightly reducing the running time from $2^{0.0473n}$ to $2^{0.0465n}$. Unfortunately this improvement is not as significant as in the FD setting. The same happens for McEliece instances with a typically small error weight way below HD, where our improvement from $2^{0.0350n}$ to $2^{0.0347n}$ is only marginal.

We suspect that the strong dependency of our algorithm on the error-weight is due to the heavy reliance on Nearest Neighbor search on every layer, which needs a sufficiently large weight ω to show its strength. We will also see this effect in the case of LPN.

LPN Problem. Let us apply our algorithm to the $LPN_{k,\tau}$ problem (Definition 3). In $LPN_{k,\tau}$ we have to solve a (n, k, ω) -decoding problem with expected weight $\omega = \tau n$ and fixed k . However, we are free to choose the number of samples n , and can therefore make the code rate $\frac{k}{n}$ arbitrarily small. Thus, for every fixed instance (k, τ) we minimize the running time $T(n, k, \tau)$ of our decoding algorithm over all n . The optimal number of samples for our algorithm for the cryptographically popular $LPN_{512, \frac{1}{4}}$ -instances is $n \approx 140.000$.

In Fig. 7, we compare different decoding algorithms for directly attacking $LPN_{512, \frac{1}{4}}$, where we suppress polynomial overheads. Here BJMM-NN would take 2^{180} steps, our algorithm has complexity 2^{169} .

m	[BM17b]		Our algorithm		
	$\log(T)/n$	$\log(S)/n$	$\log(T)/n$	$\log(S)/n$	
2	0.1003	0.0781	0.0982	0.0717	(FD)
3	0.0967	0.0879	0.0926	0.0647	
4	0.0953	0.0915	0.0885	0.0736	
2	0.0491	0.0309	0.0488	0.0290	(HD)
3	0.0473	0.0363	0.0478	0.0290	
4	0.0473	0.0351	0.0465	0.0294	
2	0.0362	0.0264	0.0360	0.0260	(McEliece)
3	0.0350	0.0280	0.0360	0.0252	
4	0.0350	0.0280	0.0347	0.0251	

Fig. 6. Running time and memory consumption of our algorithm compared to the optimized BJMM-NN variant of [BM17b].

It is however important to stress that stand-alone decoding is not the best way to attack LPN instances. As shown by Esser et al. [EKM17] a combination of the BKW algorithm [GJL14] and decoding algorithm is due to its flexible memory requirements currently the best way to tackle LPN instances in practice. Here, one first uses BKW to turn $\text{LPN}_{k,\tau}$ instances into $\text{LPN}_{k',\tau'}$ instances with reduced dimension $k' < k$ at the cost of increased error $\tau' > \tau$. Then in a second step, $\text{LPN}_{k',\tau'}$ is solved via decoding.

	$\text{LPN}_{512, \frac{1}{4}}$		$\text{LPN}_{117, \frac{255}{512}}$	
	$\log(T)$	$\log(S)$	$\log(T)$	$\log(S)$
Prange [Pra62]	213	-	117	-
BJMM [BJMM12]	190	114	117	62
BJMM-NN [MO15]	180	122	117	64
Our algorithm	169	138	75	47

Fig. 7. Complexities of different decoding algorithms for LPN instances.

Since our decoding algorithm shows its strength for large errors τ' , it seems like a perfect choice in such a hybrid BKW-decoding algorithm. In a typical attack on $\text{LPN}_{512, \frac{1}{4}}$, like the ones described in [EKM17], BKW would turn $\text{LPN}_{512, \frac{1}{4}}$ into $\text{LPN}_{117, \frac{255}{512}}$ instances, which are subsequently decoded. The calculations in Fig. 7 give us good indication that such instances with large error τ' close to $\frac{1}{2}$ can be much faster decoded by our new algorithm. However, the full extent of our improvement has yet to be determined by real experiments.

Figure 8 shows the asymptotic behavior of our algorithm on LPN-instances for varying weights τ , which also illustrates the strength of our algorithm in the high error regime. Notice that the graph of our new algorithm's complexity can be very well approximated by a line, which yields the simple formula

$$T_{LPN}(k, \tau) = 2^{1.3k\tau}.$$

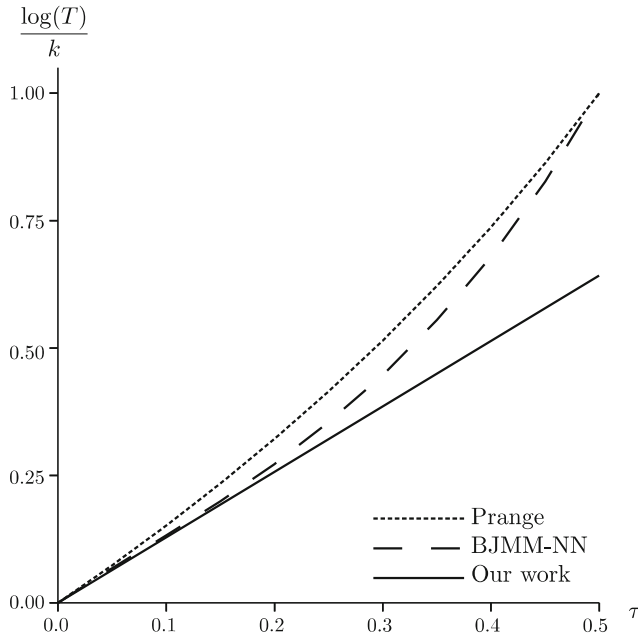


Fig. 8. Dependence on LPN error τ of [Pra62,MO15] and our algorithm.

References

- Ale03 Alekhovich, M.: More on average case vs approximation complexity. In: 44th FOCS, pp. 298–307. IEEE Computer Society Press, October 2003
- BJMM12 Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: how $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_31
- BLP08 Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 31–46. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88403-3_3
- BLP11 Bernstein, D.J., Lange, T., Peters, C.: Smaller decoding exponents: ball-collision decoding. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 743–760. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_42
- BM17a Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security (full version). Cryptology ePrint Archive: Report 2017/1139 (2017)
- BM17b Both, L., May, A.: Optimizing BJMM with nearest neighbors: full decoding in $2^{2n/21}$ and McEliece security. In: International Workshop on Coding and Cryptography (WCC 2017) (2017)

- Dum91 Dumer, I.: On minimum distance decoding of linear codes. In: Proceedings of the 5th Joint Soviet-Swedish International Workshop on Information Theory, pp. 50–52 (1991)
- EKM17 Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 486–514. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_17
- GJL14 Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 1–20. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_1
- McE78 McEliece, R.J.: A public-key system based on algebraic coding theory. Deep Space Network Progress Report 44, pp. 114–116. Jet Propulsion Laboratory, California Institute of Technology (1978)
- MMT11 May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_6
- MO15 May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 203–228. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_9
- NIS NIST Evaluation Criteria. <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography>. Accessed 24 Nov 2017
- Pra62 Prange, E.: The use of information sets in decoding cyclic codes. IRE Trans. Inf. Theory **8**(5), 5–9 (1962)
- Reg05 Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 84–93. ACM Press, New York (2005)
- Ste88 Stern, J.: A method for finding codewords of small weight. In: Cohen, G., Wolfmann, J. (eds.) Coding Theory 1988. LNCS, vol. 388, pp. 106–113. Springer, Heidelberg (1989). <https://doi.org/10.1007/BFb0019850>



QC-MDPC: A Timing Attack and a CCA2 KEM

Edward Eaton^{1(✉)}, Matthieu Lequesne^{2,3}, Alex Parent¹, and Nicolas Sendrier³

¹ ISARA Corporation, Waterloo, Canada

{ted.eaton,alex.parent}@isara.com

² Sorbonne Universités, UPMC Univ Paris 06, Paris, France

³ Inria, Paris, France

{matthieu.lequesne,nicolas.sendrier}@inria.fr

Abstract. In 2013, Misoczki, Tillich, Sendrier and Barreto proposed a variant of the McEliece cryptosystem based on quasi-cyclic moderate-density parity-check (QC-MDPC) codes. This proposal uses an iterative bit-flipping algorithm in its decryption procedure. Such algorithms fail with a small probability.

At Asiacrypt 2016, Guo, Johansson and Stankovski (GJS) exploited these failures to perform a key recovery attack. They introduced the notion of the *distance spectrum* of a sparse vector and showed that the knowledge of the spectrum is enough to find the vector. By observing many failing plaintexts they recovered the distance spectrum of the QC-MDPC secret key.

In this work, we explore the underlying causes of this attack, ways in which it can be improved, and how it can be mitigated.

We prove that correlations between the spectrum of the key and the spectrum of the error induce a bias on the distribution of the syndrome weight. Hence, the syndrome weight is the fundamental quantity from which secret information leaks. Assuming a side-channel allows the observation of the syndrome weight, we are able to perform a key-recovery attack, which has the advantage of exploiting all known plaintexts, not only those leading to a decryption failure. Based on this study, we derive a timing attack. It performs well on most decoding algorithms, even on the recent variants where the decryption failure rate is low, a case which is more challenging to the GJS attack. To our knowledge, this is the first timing attack on a QC-MDPC scheme.

Finally, we show how to construct a new KEM, called ParQ that can reduce the decryption failure rate to a level negligible in the security parameter, without altering the QC-MDPC parameters. This is done through repeated encryption. We formally prove the IND-CCA2 security of ParQ, in a model that considers decoding failures. This KEM offers smaller key sizes and is suitable for purposes where the public key is used statically.

N. Sendrier—Supported in part by the Commission of the European Communities through the Horizon 2020 program under project number 645622 (PQCRYPTO).

Keywords: Post-Quantum Cryptography
 Code-based cryptography · QC-MDPC codes · Side-channel attack
 Timing attack · CCA2 security · Key encapsulation

1 Introduction

Code-based cryptography is almost as mature as public-key cryptography itself, dating back to 1978 with the invention of the original McEliece public-key encryption scheme [28]. This scheme, when used with (as originally proposed) binary Goppa codes, has largely resisted all cryptanalytic efforts, from both classical and quantum adversaries. Because of this, code-based cryptography is a strong candidate for post-quantum standardisation, with several variants [30,31] attempting to make improvements or refinements on the original design.

Following [4,18], a new variant was proposed in 2013 using quasi-cyclic (QC) moderate density parity-check (MDPC) codes [29]. QC-MDPC codes use much shorter keys (about 10 kbits). This choice appears promising and the QC-MDPC scheme was recommended for further study by the report “Initial Recommendation of long-term secure post-quantum systems” of the European project PQCRYPTO [3]. Some hardware implementations of this scheme were published in [22,27].

The decryption algorithm of the QC-MDPC scheme is a variant of Gallager’s bit-flipping algorithm [19]. It is an iterative algorithm with a simple structure, very easy to implement, even on constrained devices. It has an inconvenient though, it is subject to failure with non-negligible probability. The algorithm proposed in the original paper [29] has a *decoding failure rate* (DFR) of 10^{-7} .

While decoding errors may not represent a serious reliability issue, in a recent paper by Guo, Johansson and Stankovski (GJS) [20], the authors showed that these decoding failures actually do represent a very serious security issue. The authors exploited this DFR and managed to successfully recover the key by analyzing the error patterns that made the decryption fail. They found that these error patterns are correlated with the key. They introduce a new tool, the *distance spectrum*, to describe the correlation. They successfully use this correlation to perform their attack and give some hints on the reason why error patterns correlated in such a way are more prone to cause decryption failure.

The original QC-MDPC primitive is extremely vulnerable because the adversary may choose the error and even force a higher weight, in this case the attack of [20] recovers the key within minutes, when attacking a parameter set intended for the 80-bit classical security level. With a semantically secure conversion (CCA security, as in [23]) it requires $2^{39.7}$ operations.

1.1 Our Contributions

In this paper we extend the analysis of the GJS attack on QC-MDPC. The GJS attack works because the decoding failure depends of the existence of common values in the spectrums of the error pattern and of the secret key. In Sect. 3 we

show that this correlation can be observed through the weight distribution of the first syndrome computed by the MDPC decoder. Pushing the analysis further we are able to quantify this bias. This allows us to perform a side-channel attack using the syndrome weight to recover the distance spectrum of the secret key. We show that the number of samples we need to make this attack work is consistent with the Chernoff bound applied to the above mentioned bias. This opens the way to theoretical estimates for the cost of attacks related to the secret key distance spectrum recovery.

Next, by remarking that the syndrome weight is correlated to the decoding time, we perform a GJS type of attack by counting the number of iterations. This provides a timing attack which is very generic and can be applied to any variant of the bit flipping algorithm which is not protected against timing attacks. Moreover, it works regardless of the failure rate. To our knowledge, this is the first timing attack on this kind of scheme. This confirms a conjecture made by Maurich and Güneysu [25] that the number of iterations in the decoding procedure leaks secret information.

In Sect. 4 we demonstrate the power of this attack by showing experimental results of the timing attack on various parameter sets and decoding procedures. This shows that the attack is practical even against the 256-bit classical parameter set. Additionally, we analyze and discuss how some other variations in the decoding procedure proposed in [27] affect the attack and its effectiveness.

Finally in Sect. 5, we show a new construction for a QC-MDPC-based KEM, called ParQ. This KEM uses QC-MDPC encryption as the underlying primitive, and does not need to alter the parameter set of the primitive itself. The scheme works by creating multiple independent encapsulations of the same key, so that a decapsulation failure only occurs if a decoding failure happens for each ciphertext. This causes the decapsulation algorithm to only fail with negligible probability, and so it entirely eliminates the possibility of using decoding failures to recover the key with the GJS attack. This scheme does not increase key sizes at all, and only increases the size of the encapsulation by a small factor (3–12×). We provide a comprehensive proof of IND-CCA2 security of the scheme, and analyse the KEM compared with other code-based key transport methods. Our proof considers the possibility of decoding failures. Other CCA2 constructions [23, 26] did not consider this, which is why the GJS attack was able to break CCA2 security. Most commentary on mitigating the GJS attack has focused on either altering the parameters of QC-MDPC to decrease the DFR or using the keys ephemerally. Through our scheme we show that there is a third option that can address decoding failures at the protocol level.

1.2 Related Work

The McEliece cryptosystem was originally proposed in [28], and low density parity-check codes were proposed in [19]. The QC-MDPC variant of McEliece was proposed in [29]. The key-recovery reaction attack we focus on in this paper was shown in [20]. In [17], the authors analyzed how the observations from [20] applied to the case of LDPC McEliece [30], showing that the attack also worked on soft decision decoding procedures.

Since the first publication of the QC-MDPC scheme, efforts have been made to tune the decoding algorithm, especially exploring the different ways to fix the thresholds in order to reduce the DFR [10]. This is discussed in details in Sect. 2.2.

Side-channel timing attacks [24] on McEliece systems other than QC-MDPC have been considered for example in [33–35] which has motivated the need for constant-time implementations [7, 13]. In [11, 12, 25], the authors demonstrated several power-analysis side-channel attacks on QC-MDPC, and [25] conjectured that it might be possible that the number of decoding rounds leaks secret information. To our knowledge, our paper is the first to conclusively show that this is in fact the case.

CCA2 conversions for McEliece systems have been considered before, most notably in [23]. General conversions for designing CCA2 KEMs from OW-CPA systems were studied in [15]. Other key exchange and key encapsulation schemes related to QC-MDPC include [5, 13, 14, 26].

A line of constructions beginning with [32], and applied to McEliece in several follow-up works [16, 36] explored the concept of the k -repetition paradigm for encryption. This paradigm bears some resemblance to our parallel KEM in Sect. 5, although these constructions are different and have a different goal: CCA2 security without random oracles.

2 QC-MDPC McEliece and the GJS Attack

2.1 Quasicyclic Moderate Density Parity Check McEliece

QC-MDPC-McEliece is a public key encryption method consisting of three algorithms. It is defined by four parameters, n , k , w , and t . The key generation algorithm `QCMDPC.KeyGen` constructs an (n, k) -linear quasicyclic code, consisting of a generator matrix G (the public key) and a parity check matrix H (the secret key), for which each row has weight w . The encryption algorithm `QCMDPC.Enc` encrypts a plaintext $x \in \mathbb{F}_2^k$ by calculating the corresponding codeword to x , xG and adding an error e of weight t to obtain the ciphertext $c = e + xG$. The decryption algorithm `QCMDPC.Dec` decodes c back to xG and recovers x .

While QC-MDPC can allow for k to be any divisor of n , we will consider the case of $n/k = 2$. We let \mathbb{E} denote the set of $e \in \mathbb{F}_2^k$ with Hamming weight t . Note that the size of each block, $r = (n - k) = k$.

Algorithm 1. QCMDPC.KeyGen

Input: Security parameter 1^λ .

Output: Public key pk , secret key sk .

- 1: Generate $h_0, h_1 \in \mathbb{F}_2^k$, both with weight $w/2$.
 - 2: Let $H = [H_0|H_1]$, where H_0 and H_1 are $k \times k$ matrices generated from h_0 and h_1 by cyclically rotating them.
 - 3: Set $G = [I_k|Q]$, where I_k is the $k \times k$ identity matrix, $Q = (H_1^{-1}H_0)^T$.
 - 4: **return** $pk = q$, the first row of Q and $sk = h_0, h_1$. These allow for the reconstruction of G and H .
-

Algorithm 2. QCMDPC.Enc**Input:** Public key $pk = g$, plaintext $x \in \mathbb{F}_2^k$, error vector $e \in \mathbb{E}$.**Output:** Ciphertext $c \in \mathbb{F}_2^n$.

-
- 1: Reconstruct $G = [I_k|Q]$ by cyclically rotating g to obtain Q .
 - 2: **return** $c = e + xG$.
-

Algorithm 3. QCMDPC.Dec**Input:** Secret key sk , public key pk , and ciphertext $c \in \mathbb{F}_2^n$.**Output:** Plaintext $x \in \mathbb{F}_2^k$ and error vector $e \in \mathbb{E}$, or decryption failure symbol \perp .

-
- 1: Reconstruct parity-check matrix $H = [H_0|H_1]$, and generator matrix $G = [I_k|Q]$.
 - 2: Run the decoding procedure on c with parity-check matrix H to recover codeword xG . If a decoding failure occurs, return \perp .
 - 3: Recover x from the first k bits of xG .
 - 4: Recover $e = c - xG$.
 - 5: **return** (x, e) .
-

Multiple parameter sets for QC-MDPC have been proposed for multiple security levels. Our interest is in the 80-bit and 256-bit classical security sets (which corresponds to at least 40-bit and 128-bit quantum security) that were originally proposed in [29], and have been further discussed in [5].

Classical bit-strength	n	k	w	t
80	9602	4801	90	84
128	20 326	10 163	142	134
256	65 542	32 771	274	264

2.2 QC-MDPC Decoding Procedure

The original paper on MDPC codes [29] proposes to use a hard decision version of Gallager's bit-flipping algorithm for decoding LDPC codes [19]. The main idea is the following. At each iteration, the algorithm computes the number of unsatisfied parity-check equations associated to each bit. Each bit that is involved in $\geq b$ unsatisfied equations is flipped, for b some threshold, and the syndrome is recomputed. This repeats until the syndrome becomes zero. In practice, the algorithm stops after fixed number of iterations and this is considered a decoding failure.

For our main analyses we use decoder \mathcal{D}_1 from [27] with fixed thresholds $\{95, 85, 80, 76, 74, 73, 72, 72\}$. \mathcal{D}_1 is a modification of Gallager's algorithm which updates the syndrome in place after each bit flipped. Algorithm 4 is the normal out-of-place bit flipping algorithm and Algorithm 5 is the in-place version.

Algorithm 4. Iterative bit flipping decoding algorithm

Input: $c = (c_0, \dots, c_{n-1}) \in \mathbb{F}_2^n$, $H = (h^{(0)}, \dots, h^{(n-1)}) \in \mathbb{F}_2^{r \times n}$
 $s \leftarrow H \cdot c^\top$ ▷ compute the syndrome
while $s \neq 0$ **do**
 for $i = 0, \dots, n - 1$ **do**
 if $\langle s, h^{(i)} \rangle \geq b$ **then** ▷ if number of unsatisfied equations \geq threshold b
 $c_i \leftarrow c_i \oplus 1$ ▷ flip the i^{th} bit
 $s \leftarrow H \cdot c^\top$
return c

Algorithm 5. In-Place: Iterative bit flipping decoding algorithm

Input: $c = (c_0, \dots, c_{n-1}) \in \mathbb{F}_2^n$, $H = (h^{(0)}, \dots, h^{(n-1)}) \in \mathbb{F}_2^{r \times n}$
 $s \leftarrow H \cdot c^\top$ ▷ compute the syndrome
while $s \neq 0$ **do**
 for $i = 0, \dots, n - 1$ **do**
 if $\langle s, h^{(i)} \rangle \geq b$ **then** ▷ if number of unsatisfied equations \geq threshold b
 $c_i \leftarrow c_i \oplus 1$ ▷ flip the i^{th} bit
 $s \leftarrow s \oplus h^{(i)}$
return c

Variable thresholds. A more recent approach, studied in [10], is to choose the values of b at each iteration depending on the syndrome weight at the time. This approach gives the best results so far, both in terms of decryption failure rate and average number of iterations.

In both cases, until now the thresholds were claimed as experimental results with no explanation on the way they were generated. In Appendix B we discuss a procedure to obtain such thresholds for any security parameters.

2.3 The GJS Attack

The key recovery attack in [20] is a reaction attack. It takes advantage of the decoding failures that occasionally occur during decryption. It assumes only that an adversary is able to tell when such an error has occurred, for example because a request for resend is sent back. It consists of two steps. The first step is to calculate the *distance spectrum* of the secret key (or one part of the secret key), based on observing a large number of error vectors that resulted in a decoding failure. The second step is to reconstruct the secret key based on its distance spectrum.

In this paper, we will focus our attention on the first step. Reconstructing the secret key from the distance spectrum has been analysed before [17, 20], and shown to be fairly fast and simple as compared to the first step, and is an entirely offline computation, requiring no communication.

Definition 1 (Distance Spectrum). *The distance spectrum of a vector $h \in \mathbb{F}_2^r$, denoted $\Delta(h)$, is the set of distances δ such that there exist two non-zero bits of h at distance δ . The distance are counted cyclically.*

$$\Delta(h) = \left\{ \delta : 1 \leq \delta \leq \left\lfloor \frac{r}{2} \right\rfloor, \exists(i, j), \begin{array}{l} 0 \leq i < j < r, \\ h[i] = h[j] = 1, \\ \min\{j - i, r - (j - i)\} = \delta \end{array} \right\}$$

where $h[i]$ denotes the i^{th} entry of the binary vector h .

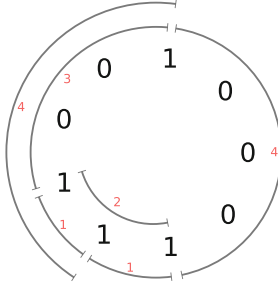


Fig. 1. Distance spectrum of 100100011_2

For example, the distance spectrum of the vector 100100011_2 is $\{1, 2, 3, 4\}$ (Fig. 1). Note that any cyclic shift or reversal of a vector will result in the same distance spectrum. In [17, 20], it was shown how to quickly reconstruct a vector (up to a reversal or cyclic shift) from a distance spectrum. The first step of the GJS attack is to find the distance spectrum of the first half h_0 of the secret key (h_0, h_1) . From this, h_0 can be computed, which allows us to also calculate h_1 by elementary linear algebra.

In order to analyse more precisely the results, we need to take into account the fact that some distances may appear more than once.

Definition 2 (Distance Spectrum with multiplicity). *The distance spectrum with multiplicity of a vector $h \in \mathbb{F}_2^r$, denoted $\Delta^\mu(h)$, is a vector of $\mathbb{N}^{\lfloor \frac{r}{2} \rfloor}$ such that for every distance $1 \leq \delta \leq \lfloor \frac{r}{2} \rfloor$, its δ^{th} component $\Delta^\mu(h)[\delta]$ is the number of existing sets of two non-zero bits of h at distance δ . The distance are counted cyclically.*

Example 1. For $h = 0011000011_2$ (see Fig. 1), then $\Delta^\mu(h) = [2, 1, 1, 2]$.

In general we can see that if a vector $h_0 \in \mathbb{F}_2^k$ has weight w_0 , then the distance spectrum with multiplicity of h_0 will be a vector of size $\lfloor k/2 \rfloor$ such that the sum of the entries of $\Delta^\mu(h_0)$ is $\binom{w_0}{2}$.

Finding the distance spectrum of the secret key is done by taking note whether a decoding failure occurs for a large number of error vectors. This is done because of the following observation:

Observation 1 (GJS, Key Observation). When a distance in the error vector used in a QC-MDPC encryption matches a distance in the distance spectrum of the secret key, a decoding failure is *less* likely to occur.

Based on this observation, it was noticed that by carefully calculating the decoding failure rate for errors that have a given distance vs. those that do not, the multiplicity of that distance in the secret key’s distance spectrum can be correctly guessed. Note that this observation applies to each half of the error vector (and parity check matrix) independently. So when we refer to the distance spectrum of the error or parity-check matrix, we mean the distance spectrum of the first k bits, unless stated otherwise.

Algorithm 6 was proposed in [20] for attacking the CCA security of a QC-MDPC implementation.

Algorithm 6. GJS CCA attack

```

1: Initialize  $observed_d = 0$  and  $failed_d = 0$  for  $d \in \{1, \dots, \lfloor k/2 \rfloor\}$ .
2: for  $i = 1$  to  $M$  do
3:   Send  $c = \text{QCMDPC.Enc}(x, e)$  with a uniformly random  $e = [e_0 || e_1]$  to target.
4:   for  $d \in \Delta(e_0)$  do
5:     Increment  $observed_d$  by 1.
6:     if Decoding failed for  $c$  then
7:       Increment  $failed_d$  by 1.
8: return  $failed_d/observed_d$  for  $d \in \{1, \dots, \lfloor k/2 \rfloor\}$ .

```

The resulting values, $failed_d/observed_d$ for each d give an estimate of the decoding failure rate for error vectors with d in their distance spectrum. We can then recover the distance spectrum, identifying the multiplicity of each distance from the following observation:

Observation 2 (GJS). For a fixed key, the decoding failure rate for error vectors with d in their distance spectrum is inversely proportional to the multiplicity of d in the distance spectrum of the key.

For large enough values of M , the decoding failure rate clearly separates into bands. These bands exactly correspond to the multiplicity of that distance in $\Delta(h_0)$. This allows an attacker to recover $\Delta(h_0)$, and thus the secret key.

The complexity of the attack is dominated by the value M . The decoding failure rates for different multiplicities are quite close together, and so a very accurate estimation is need in order to properly decide on the multiplicity. In [20], the authors found that $M = 2^{29}$ was sufficient for the 80-bit classical parameter set, using the Gallager decoding algorithm. They conjectured that using a more sophisticated decoding algorithm like that in [29], would mean that M would have to be increased by an amount proportional to the difference in the decoding failure rate. They also conjectured that higher parameter sets would not significantly alter the effectiveness of the attack, as the decoding failure rate does not significantly change.

3 Analysis and Timing Attack

3.1 Correlation

Our attack is based on the fact that the average syndrome weight is slightly different if the relative position of non-zero bits in the key and the error are correlated.

For the sake of simplicity, in this section, we will consider a parity-check matrix made of one single circulant block in $H \in \mathbb{F}_2^{k \times k}$ instead of two. We will see later that the practical results are the same. We denote by $h \in \mathbb{F}_2^k$ the first row of the matrix H . The variable t still represents the weight of the error e , so here the numerical value of t should be half its usual value.

Without any information. Let us suppose that we do not have any information on the key. For a random key vector h of size k and weight d and a random error vector e of size k and weight t , denote by $f(k, d, t, b)$ the probability that the scalar product in \mathbb{F}_2 is of parity b :

$$f(k, d, t, b) := \Pr[\langle h, e \rangle = b] = \sum_{i=0, i \text{ is of parity } b}^d \frac{\binom{d}{i} \binom{k-d}{t-i}}{\binom{k}{t}}.$$

The average syndrome weight of an error e and a parity-check matrix generated by cyclic shifts of h is k times the probability that a bit is non-zero (see [9, page 91]), that is:

$$\mathbb{E}[\text{wt}(H \cdot e^\top)] = k \cdot f(k, d, t, 1).$$

Case of two consecutive non-zero bits in the key. Now, suppose the key vector h has ℓ times two consecutive non-zero bits. Let us observe the shifts of the vector:

shift(h) =	1 1	$u, \text{wt}(u) = d - 2$	ℓ times
shift(h) =	1 0	$u, \text{wt}(u) = d - 1$	$d - \ell$ times
shift(h) =	0 1	$u, \text{wt}(u) = d - 1$	$d - \ell$ times
shift(h) =	0 0	$u, \text{wt}(u) = d$	$k - 2d + \ell$ times.

Suppose that the first two bits of the error vector are non-zero, that is:

$$e = \left[\begin{array}{|c|c|} \hline 1 & 1 \\ \hline \end{array} \quad u, \text{wt}(u) = t - 2 \right].$$

With this extra assumption on the form of h and e , the average syndrome weight of e with respect to the the parity-check matrix H generated by cyclic shifts of h can now be approximated by:

$$\begin{aligned} \mathbb{E}[\text{wt}(H \cdot e^\top)] = & \ell f(k - 2, d - 2, t - 2, 1) \\ & + 2(d - \ell) f(k - 2, d - 1, t - 2, 0) \\ & + (k - 2d + \ell) f(k - 2, d, t - 2, 1). \end{aligned} \tag{1}$$

Contrary to the previous result, this is an approximation. Indeed, this model assumes that the rest of the vector (denoted by u) is random for each shift. It does not take into account the covariance between the bits of the syndrome. Previously we were averaging on all the lines and the covariance was therefore null, while here the fact that we group the rows depending on the value of the first two bits breaks the symmetry. Still, we will see that the approximation is close to the real value and we can neglect the correction term for the rest of the study.

Exploiting the leak. Suppose that we only consider error patterns starting with two consecutive non-zero bits, the syndrome weight is expected to be slightly different on average, depending on ℓ the number of times two consecutive bits are non-zero in the key vector h . Moreover, the expected value varies linearly with ℓ . Therefore, if we observe enough values of the syndrome weight, we can recover the value of ℓ .

Definition 3 (Average syndrome weight with multiplicity). *Let us denote by D_ℓ the following set:*

$$D_\ell := \left\{ (h, e) \in \mathbb{F}_2^k \times \mathbb{F}_2^k \mid \text{wt}(h) = d, \text{wt}(e) = t, \delta \in \Delta(e), \Delta^\mu(h)[\delta] = \ell \right\}.$$

The average syndrome weight with multiplicity $\bar{\sigma}_\ell$ is the expectation of the syndrome weight for a uniform distribution of (h, e) over D_ℓ :

$$\bar{\sigma}_\ell := \mathbb{E}_{(h,e) \sim \mathcal{U}(D_\ell)} [\text{wt}(H \cdot e^\top)].$$

From the Eq. (1) in Sect. 3.1 we know that we can approximate $\bar{\sigma}_\ell$ by:

$$\begin{aligned} \bar{\sigma}_\ell = & \ell \quad f(k-2, d-2, t-2, 1) \\ & + 2(d-\ell) \quad f(k-2, d-1, t-2, 0) \\ & + (k-2d+\ell) \quad f(k-2, d, t-2, 1). \end{aligned}$$

$$\text{with } f(k, d, t, b) := \sum_{i=0, i \text{ is of parity } b}^d \frac{\binom{d}{i} \binom{k-d}{t-i}}{\binom{k}{t}}$$

Comparison with measured values. The values of $\bar{\sigma}_\ell$ correspond to the different clusters that we can see on the figures. According to the approximation, the value of $\bar{\sigma}_\ell$ is linear in the multiplicity: $\bar{\sigma}_0 - \bar{\sigma}_\ell = \ell \cdot (\bar{\sigma}_0 - \bar{\sigma}_1)$. This is consistent with what we observe.

With the usual parameters for 80-bit security, (here using $t = 42$ as there is only one block) we obtain $\bar{\sigma}_0 = 1324.23$ and $\bar{\sigma}_1 = 1323.28$.

When comparing the values to those measured on Fig. 2, we can see that the measured $\bar{\sigma}_0$ is slightly lower than the approximated value, and on the contrary $\bar{\sigma}_1$ is slightly higher. This error is due to the approximation that neglects the covariance. When performing the same experiment on parameters for LDPC codes, where the covariance is much smaller, the measures correspond exactly to the computed values.

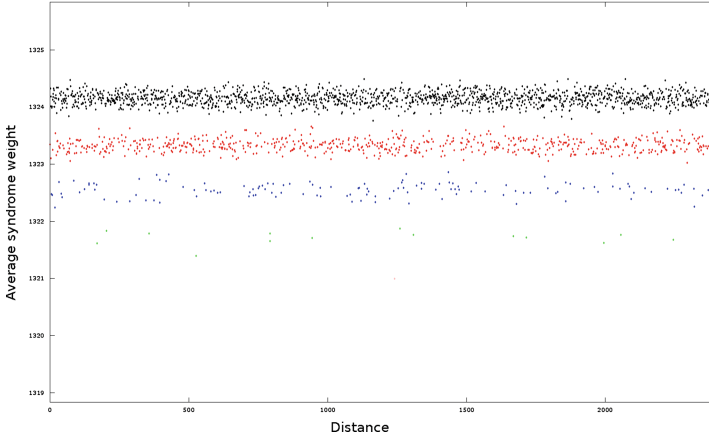


Fig. 2. Attack on the syndrome weight (1 block): average syndrome weight per distance, 10^5 samples. The color of the distances indicate their multiplicity in the key spectrum (black = 0, red = 1, blue = 2, green = 3) (Color figure online)

As a consequence, the real distance $\bar{\sigma}_0 - \bar{\sigma}_1$ is smaller than the one computed using Eq. (1). Hence, the theoretical analysis gives an interesting bound on the relative distance $\varepsilon = \frac{\bar{\sigma}_0 - \bar{\sigma}_1}{k}$: $\varepsilon_{\text{measured}} < \varepsilon_{\text{computed}}$.

Hypothesis testing. Each syndrome is the result of k scalar products between the error and a parity-check equation. When the error contains a distance present in the spectrum of the key with multiplicity ℓ , the average syndrome weight is $\bar{\sigma}_\ell$, this means that on average $\bar{\sigma}_\ell$ of the k parity-check equations are not verified. Hence, under the independence assumption, we can see each bit of the syndrome as a Bernoulli trial satisfied with probability $\frac{\bar{\sigma}_\ell}{k}$.

Here, our goal is to decide for each distance δ whether or not δ is in the distance spectrum of h . We do not care about the multiplicity. Formally, we want to distinguish D_0 from $\cup_{\ell \geq 1} D_\ell$. Let us by $D_{\geq 1} := \cup_{\ell \geq 1} D_\ell$. We can define $\bar{\sigma}_{\geq 1}$ on $D_{\geq 1}$ just like we defined $\bar{\sigma}_\ell$ on D_ℓ . The sets are disjoint so we have
$$\bar{\sigma}_{\geq 1} = \frac{\sum_{\ell \geq 1} \bar{\sigma}_\ell |D_\ell|}{\sum_{\ell \geq 1} |D_\ell|}.$$

Hence, deciding whether a distance is in the spectrum of the key or not is just like distinguishing a random binary variable with success probability $p_0 := \bar{\sigma}_0$ from a random binary variable with success probability $p_1 := \bar{\sigma}_{\geq 1}$. This is a classic problem of hypothesis testing.

Note that for our parameters, the size of D_ℓ for $\ell \geq 2$ is negligible compared to D_1 , hence there is no practical need to distinguish $\bar{\sigma}_1$ from $\bar{\sigma}_{\geq 1}$.

Sample size. There is a lot of literature about hypothesis testing, and in particular a theorem from Chernoff [21] concerning such cases.

Proposition 1 (Chernoff’s bound). *Let $0 < p < 1$, let X_1, X_2, \dots, X_N be independent binary random variables, with $\Pr[X_k = 1] = p$ and let $S_N = \frac{\sum_{k=1}^N X_k}{N}$. Then for any $t \geq 0$,*

$$\Pr[|S_N - p| \geq t] \leq 2e^{-2Nt^2}.$$

This can be used to understand how the number of samples required to find the key evolves. Here we want to distinguish p_0 from p_1 , we will use $\frac{p_0+p_1}{2}$ as the decision threshold. Chernoff’s bound states that we should have $N \sim \frac{1}{\varepsilon^2}$ repeated Bernoulli trials for the decision test to be relevant, where $\varepsilon = |p_1 - p_0| = \frac{\bar{\sigma}_0 - \bar{\sigma}_1}{k}$ is the distance between the two outcomes.

To decide whether a particular distance δ is in the spectrum or not, we need to compute the mean of N Bernoulli trials, but each syndrome weight is already the sum of the results of k Bernoulli tests. Hence, we need to observe the weight of $\frac{N}{k}$ syndromes. These syndromes need to be in one of the D_ℓ , this means that the distance δ needs to be in the spectrum of the error pattern that generates the syndrome. As the error patterns are generated uniformly, we proceed by rejection sampling to ensure this condition. The number of vectors of size k and weight w that do not contain a particular distance is $\prod_{j=0}^{w-1} (k-3j)$, so neglecting the cases of multiplicity we obtain a good approximation of the frequency of such vectors with:

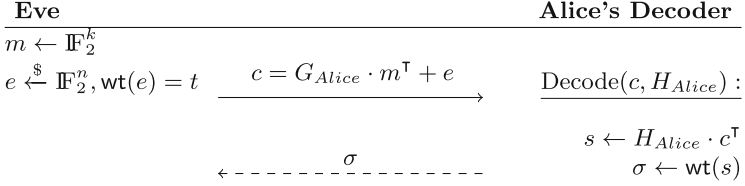
$$\alpha := \Pr(\delta \in \Delta(e)) \approx 1 - \frac{\prod_{j=0}^{\lfloor \frac{t}{3} \rfloor - 1} (k-3j)}{\prod_{j=0}^{\lfloor \frac{t}{2} \rfloor - 1} (k-j)}.$$

Hence, to decide whether or not $\delta \in \Delta(h)$, we need to observe the decoding of $\frac{N}{\alpha \cdot k}$ syndromes, with $N \sim \frac{1}{\varepsilon^2}$. As we use the same data to decide for all distances, this is the number of samples needed to recover the whole spectrum.

3.2 Attack on the Syndrome Weight

Attack Model. The scenario for our attack is the following. Eve can encrypt random messages using the QC-MDPC scheme described in Sect. 2.1 and Alice’s public key. She has access to the plaintext but cannot choose the messages. She sends the messages for decryption. Whenever the device decodes a message sent by Eve, she has a way to observe the weight of the syndrome.

The attack we describe here is an abstraction. We do not focus on how, or even if, Eve gets access to the data. It might be possible or not depending on a particular implementation and on the abilities of the attacker. The point is to establish through a simulation that some secret information leaks from the syndrome weight and to compare the cost of that simulation with the theoretical analysis of the previous section.



We suppose that Eve's error patterns are randomly generated. Indeed, in the scheme, semantically secure conversions ensure that the error patterns are random [23]. If we allow Eve to choose the error patterns, this will only make the attack easier, as in [20].

Contrary to [20], we collect information from all the error patters, not only those leading to a decoding failure.

Attack on Syndrome Weight. Our goal is to compute the distance spectrum of Alice's private key. For each distance δ between 1 and $\lfloor \frac{k}{2} \rfloor$ we want to decide whether or not $\delta \in \Delta(h_{\text{Alice}})$. As we have seen in Sect. 3.1, for each distance $\delta \in \Delta(e)$, the expected average weight of the syndrome $\sigma = \text{wt}(s)$, where $s = H_{\text{Alice}} \cdot c^\top = H_{\text{Alice}} \cdot e^\top$, is expected to be different if $\delta \in \Delta(h_{\text{Alice}})$.

Hence, the idea is, for each distance δ , to compute the average value of the syndrome weight σ for error patterns e such that $\delta \in \Delta(e)$. The error patterns are generated randomly and each error e can be used to obtain information on all the distances in its spectrum. This leads to Algorithm 7.

Following the discussion in Sect. 3.1, we will take $\text{threshold} = \frac{\bar{\sigma}_0 + \bar{\sigma}_1}{2}$.

Algorithm 7. Computing the distance spectrum

Input: N the size of the sample, oracle access to the decoder

SyndromeCount $\leftarrow (0, \dots, 0) \in \mathbb{N}^{\lfloor \frac{k}{2} \rfloor}$

OccurenceCount $\leftarrow (0, \dots, 0) \in \mathbb{N}^{\lfloor \frac{k}{2} \rfloor}$

$\Delta \leftarrow (0, \dots, 0)$

for $0 \leq i \leq N - 1$ **do**

$e \xleftarrow{\$} \mathbb{F}_2^n, \text{wt}(e) = t$

$\sigma \leftarrow \text{OracleDecoder}(e)$

$\triangleright \sigma = \text{wt}(e \cdot H_{\text{Alice}}^\top)$

for $\delta \in \Delta(e)$ **do**

SyndromeCount[δ] += σ

OccurenceCount[δ] += 1

for $1 \leq \delta \leq \lfloor \frac{k}{2} \rfloor$ **do**

if SyndromeCount[δ]/OccurenceCount[δ] < threshold **then**

$\Delta[\delta] \leftarrow 1$

return Δ

3.3 Attack on Iteration Count

Now that we know that the syndrome weight leaks information, any parameter correlated to this quantity could be used for a side channel attack. An interesting parameter that is often easy to measure is the number of iterations of a loop.

The decoding algorithm for QC-MDPC codes is an iterative algorithm with no termination proof. The number of rounds needed to correct the errors varies. This has been studied by in [10]. As mentioned in Sect. 2.2, the algorithm depends on the way we chose the thresholds. For most instances, using fixed or variable thresholds, the algorithm usually corrects the error in 3 rounds, but some instances need 4, 5 or even more iterations. Usual implementations abort after a certain number of rounds (around 10), this is what was used for the attack in [20].

Experimentally, we observe that the correlations between the spectrum of the error and the spectrum of the key has an impact on the average decryption time. The more distances appear both in spectrum of the error and in the spectrum and the key, the fewer the number of iterations needed to decode on average. This appears clearly on Fig. 3. We note that the correlation is slightly more important on Fig. 3 when we use variable thresholds than with fixed thresholds (the average value is lower for variable thresholds, but the same scale is used for both figures).

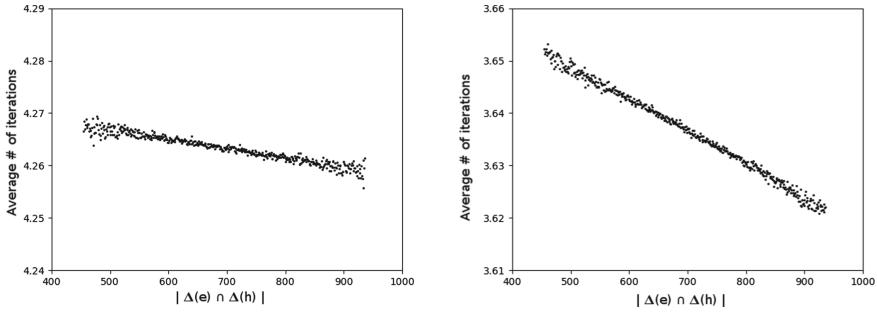


Fig. 3. Average number of iterations needed for decryption, depending on the size of the intersection of the spectrum of the error and the spectrum of the key. 2^{29} samples, 128-bit security QC-MDPC scheme, decoding with fixed thresholds (left) and variable thresholds (right). Note that use of variable thresholds results in stronger correlation.

This motivated us to try to perform a theoretical timing attack (Algorithm 8). The scenario is the same as previously, but instead of observing the syndrome weight, Eve can measure the number of iterations needed to decode her message. To obtain the spectrum, Eve uses the exact same data collection algorithm: for every distance in the spectrum, she computes the average number of iterations needed to correct an error containing this distance.

This works well and it is possible to fully recover the distance spectrum with variable thresholds using 2^{25} samples on 80-bit security QC-MDPC scheme, 2^{25} samples for 128-bit security parameters (see Fig. 6) and 2^{28} samples for 256-bit security parameters. For fixed thresholds, we manage to recover the spectrum for 256-bit security with 2^{28} samples.

Algorithm 8. Timing attack on QC-MDPC

```

1: Initialize  $observed_d = 0$  and  $iterations_d = 0$  for  $d \in \{1, \dots, \lfloor k/2 \rfloor\}$ .
2: for  $i = 1$  to  $M$  do
3:    $e \xleftarrow{\$} \mathbb{F}_2^m$ ,  $wt(e) = t$ 
4:    $c \leftarrow \text{QCMDPC.Enc}(x, e)$ 
5:   Send  $c$  to target.
6:    $n \leftarrow$  number of iterations (from side channel).
7:   for  $d \in \Delta(e_0)$  do
8:      $observed_d += 1$ .
9:      $iterations_d += n$ .
10: Return  $iterations_d/observed_d$  for  $d \in \{1, \dots, \lfloor k/2 \rfloor\}$ .

```

4 Experimental Results

Results of Syndrome Attack. The spectrum recovery algorithm was first tried on a simplified version of the scheme using only one block, in order to compare to the expected behaviour. The result is striking. Using the usual parameters for 80-bit security, with one hundred thousand samples, the spectrum appears very clearly and we can even see the multiplicities, that is, distances that appear several times in the key, see Fig. 2. When pushing to one billion samples, there is no room for confusion.

When performing the same experiment on the real QC-MDPC scheme with two blocks, we obtain similar results. The attack is performed on each block separately, that is for each error pattern, we added the syndrome weight to the counters of all distances present in the first half of the error to recover the spectrum of the first block. Because there is no correlation between the two halves of the error pattern, the presence of the second block acts as a random noise added to the syndrome weight. Hence the only difference is that we need more samples to reduce the variance and distinguish well which distances are in the key spectrum. Note that it is possible to compute the spectrum of both blocks at the same time, so there is no need to double the number of samples to recover the second block.

For 80-bit security parameters, we can see on Fig. 4 the spectrum appearing more and more distinctively when we increase the number of samples. With 2^{20} samples, we can fully distinguish the spectrum. The same attack requires 2^{23} samples for 128-bit security parameters and 2^{25} for 256-bit security parameters.

This attack was also performed when another error is added to the syndrome, like in the Ouroboros scheme [14] (with an additional error of weight $3d$). Again, this only adds random noise and we can recover the spectrum with around a few million samples for the 80-bit security parameters.

Results of iteration attack. After running Algorithm 8 we collect data corresponding to the average number of iterations it took to decode an error when d is present. The resulting plots (Fig. 5) look very similar to the plots of the decoding failure rate that result from Algorithm 6. Once the bands have completely

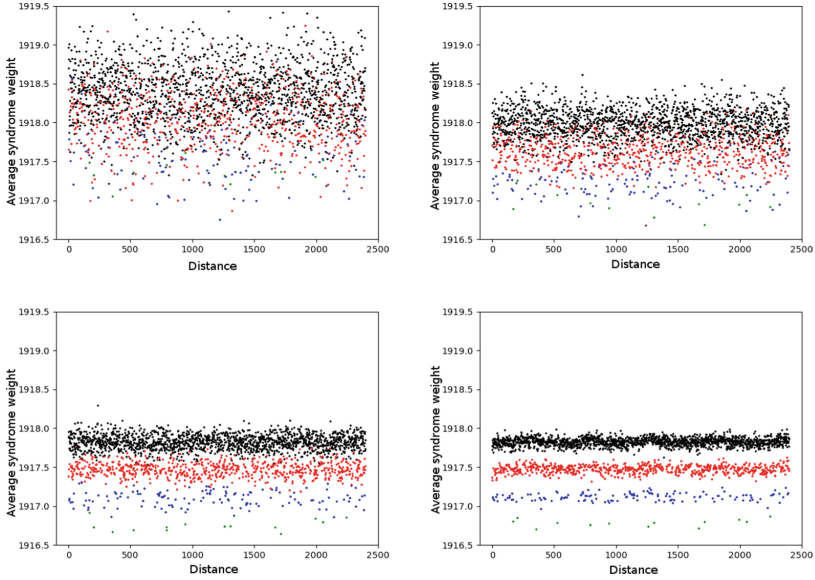


Fig. 4. Average syndrome weight per distance, (from left to right, from top to bottom) 2^{14} , 2^{16} , 2^{18} and 2^{20} samples, 80-bit security QC-MDPC scheme. The color of the distances indicate their multiplicity in the key spectrum (black = 0, red = 1, blue = 2, green = 3, purple ≥ 4) (Color figure online)

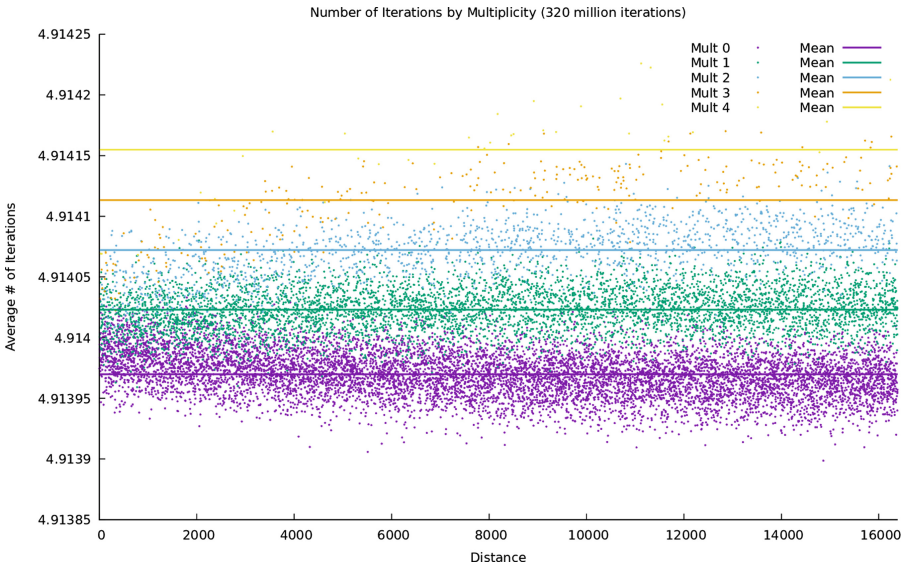


Fig. 5. Attack using the number of decoding iterations against parameters for 256-bit security with fixed threshold decoding.

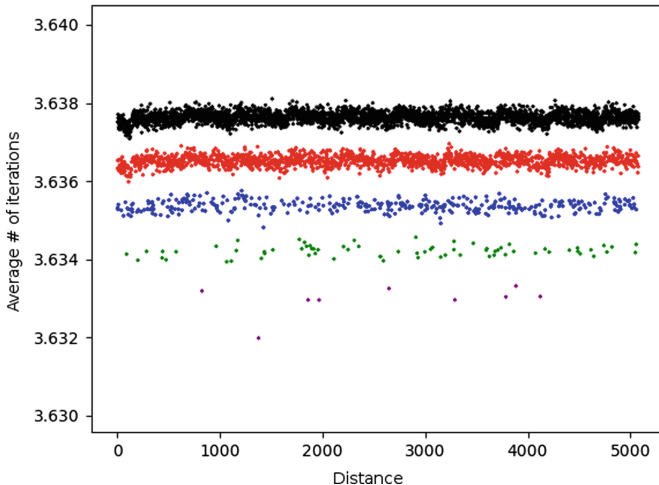


Fig. 6. Attack using the number of decoding iterations, with 2^{25} samples, against parameters for 128-bit security QC-MDPC scheme with variable threshold decoding. The color of the distances indicate their multiplicity in the key spectrum (black = 0, red = 1, blue = 2, green = 3, purple ≥ 4) (Color figure online)

separated, the distance spectrum (and thus the secret key) can be recovered in the same way it was in the GJS [20] attack.

This side-channel attack is much faster than the reaction attack. An intuitive explanation for the speedup is that differences in the number of iterations are much more common than decoding errors. This allows more information about the correlations to be collected per iteration.

4.1 In-Place Decoder vs. Out-of-Place Decoder

We observed that changes to the decoding algorithm can have a significant impact on the information gathered during the attack.

\mathcal{D}_1 uses in-place updates to the syndrome which seems to cause some asymmetry in the errors with respect to distance. For example, in Fig. 5 the bands converge as distance increases.

Postponing the updates until the end of each iteration (using \mathcal{B} from [27]) seems to eliminate this asymmetry and reduces the correlation between number of iterations and distance multiplicity. This may reduce the efficiency of the attack.

Figure 7 shows a direct comparison between these two types of decoders. Note that the relationship between number of iterations and multiplicity is inverted between decoders.

We are not sure why this is the case but give a possible explanation for the behaviour. When distances match the resulting behaviour is a decrease in total changes to counters (both correct and incorrect). As noted in [20] this decreases

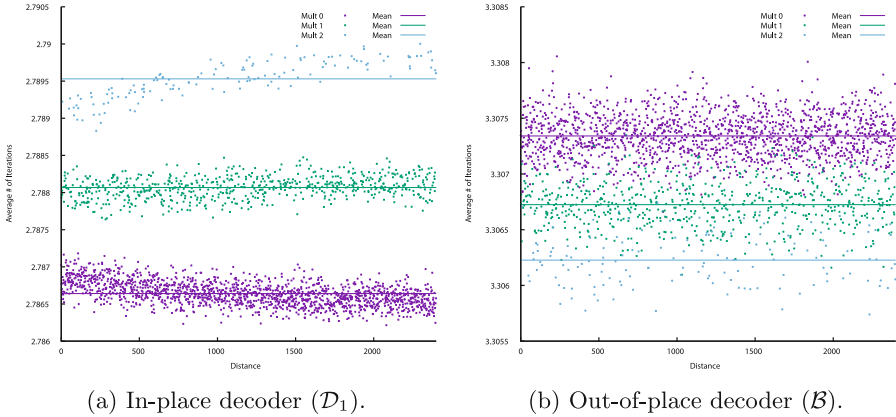


Fig. 7. Comparison of in-place and out-of-place decoders with fixed thresholds using 30 million iterations against 80-bit security. Decoder definitions are from [27].

the error rate since it decreases the probability of an incorrect change. It also decreases the expected number of bits flipped which could cause an increase in the expected number of iterations.

When multiple bits are flipped at once in the out-of-place decoder the benefit of a correct flip early in an iteration is removed so it is possible that benefit of early flipping is dominated by the increased chance of an incorrect flip.

5 Eliminating Decoding Failure Vulnerabilities

In this section we present ParQ—A KEM constructed from repeating a QC-MDPC encryption scheme in order to eliminate the effect of decoding failures. The general idea is for the ciphertext to include several independent encapsulations of the same key in such a way that the scheme achieves CCA2 security, and so that a decapsulation failure occurs in ParQ only if a decryption failure occurs in every instance of the underlying QC-MDPC scheme. As current estimates for the failure rate indicate that failures occur at a rate of roughly 2^{-23} , this suggests that a small amount of parallelization ($3\text{--}12\times$) will make decapsulation failures occur in ParQ at a negligible rate, thus removing the possibility of implementing a reaction attack based on these failures.

5.1 ParQ—A Parallelized QC-MDPC KEM

ParQ is largely characterized by the same parameters as other QC-MDPC code-based schemes, specifically, k , the plaintext length, $n = 2k$, w , the weight of the secret key, and t , the weight of the error. In addition to these parameters, ParQ has the parameter P , denoting the degree of parallelization. P must be greater than or equal to 2, and should generally be chosen to be in the range of $3 - 12$. ParQ is described by three algorithms: ParQ.KeyGen for key generation (omitted

Algorithm 9. ParQ.Enc**Input:** Public key pk , a seed $s \in \{0, 1\}^k$.**Output:** Session key K , key encapsulation $C = (c_1, \dots, c_P)$.

```

1: for  $i = 1$  to  $P$  do
2:   Let  $e_i = \text{ErrGen}(s||i)$ .
3:   Compute  $x_i = s \oplus \text{PRF}(e_i||i)$ .
4:   Compute  $c_i = \text{QCMDPC.Enc}(pk, x_i, e_i)$ .
5: Compute  $K = \text{KDF}(s)$ .
6: Return session key  $K$ , key encapsulation  $C = (c_1, \dots, c_P)$ .

```

Algorithm 10. ParQ.Dec**Input:** Secret key sk , public key pk , and encapsulation $C = (c_1, c_2, \dots, c_P)$.**Output:** Session key K , or decapsulation failure symbol \perp .

```

1: for  $i = 1$  to  $P$  do
2:   Run  $(x_i, e_i) \leftarrow \text{QCMDPC.Dec}(sk, c_i)$ .
3:   if QCMDPC.Dec successfully decoded for the first time then
4:     Set used index  $j = i$ .
5: if QCMDPC.Dec failed to decode for  $i = 1$  to  $P$  then
6:   Return decapsulation failure  $\perp$ .
7: Compute  $s = x_j \oplus \text{PRF}(e_j||j)$ .
8: Compute  $K, C' = (c'_1, c'_2, \dots, c'_P) \leftarrow \text{ParQ.Enc}(pk, s)$ .
9: if  $c_i = c'_i$  for all  $i \in \{1, \dots, P\}$  then
10:  Return  $K$ .
11: else
12:  Return decapsulation failure  $\perp$ .

```

since it is the same as Algorithm 1), ParQ.Enc for encapsulation, and ParQ.Dec for decapsulation. It uses three functions which we model as random oracles, ErrGen, PRF, and KDF, which map onto \mathbb{E} , \mathbb{F}_2^k , and $\{0, 1\}^\lambda$, respectively.

5.2 Overview of IND-CCA2 Reduction for ParQ

For the rest of this section, we show the IND-CCA2 (INDistinguishable under Chosen Ciphertext Attack) security of the ParQ KEM. We show this by reduction from the OW-CPA (One Way under Chosen Plaintext Attack) security of the QC-MDPC McEliece system. We use the standard definitions of IND-CCA2 and OW-CPA security, which can be found in Appendix A for completeness.

Theorem 1. *Let \mathcal{A} be an adversary capable of winning the IND-CCA2 security game with the ParQ KEM with q_d decapsulation queries and q_{ErrGen} , q_{PRF} , and q_{KDF} queries to the random oracles ErrGen, PRF, and KDF respectively, in time t and with advantage ϵ . Then there exists a reduction \mathcal{B} that uses \mathcal{A} as a subroutine by simulating the IND-CCA2 environment in order to break the OW-CPA security of QC-MDPC McEliece, in time $\approx t$ and with success probability $\gamma(\epsilon/P - \delta)$, where δ is negligible and γ is negligibly close to 1 in the security parameter.*

In order to establish IND-CCA2 security via a reduction from OW-CPA, we need to establish how to embed the given OW challenge c^* into an IND challenge (Sect. 5.4), and how to successfully respond to decapsulation queries (Sect. 5.5). Then we need to show that the simulation satisfies several key properties: that the simulated challenge is indistinguishable from a real challenge (Sect. 5.4), that an adversary’s ability to solve the IND challenge allows the simulation to solve the OW challenge (Sect. 5.4), and that the simulated responses to decapsulation queries are indistinguishable from actual responses to a decapsulation query (Sect. 5.5).

In ParQ, we have that $e_i = \text{ErrGen}(s||i)$ and $x_i = s \oplus \text{PRF}_i(e_i)$, or $s = x_i \oplus \text{PRF}_i(e_i)$. So for any possible c and i , there is at most one s associated with it such that $c = \text{QCMDPC.Enc}(s \oplus \text{PRF}(\text{ErrGen}(s||i)||i), \text{ErrGen}(s||i))$.

5.3 Simulating the Random Oracle

ParQ makes use of three functions that we will model as random oracles—a pseudo random function PRF, an error generation function ErrGen, and a key derivation function KDF. Each random oracle will be maintained by a standard ‘on-the-fly’ method. For each oracle, a table is maintained specifying which queries have been made and what the responses were. For each oracle, when a query is made, we first check if it has been queried before, and if so, respond with the same response made before. We then specify how to handle new queries.

For new queries to ErrGen of the form $s||i$ we choose a uniformly random error vector $e \in \mathbb{E}$. We then also calculate $x = s \oplus \text{PRF}(e||i)$ and add e and $c = \text{QCMDPC.Enc}(x, e)$ to the table. We then respond with e .

For new queries to the PRF oracle of the form $e||i$ we first check and see if e is the error vector associated with the challenge ciphertext c^* . We do this by using the generator matrix G to see if $c^* - e$ is a codeword. If so, then we have solved the challenge. Otherwise, generate a uniformly random string from $\{0, 1\}^k$, add it to the table and respond.

New queries to KDF can simply be handled by responding with a uniformly random $\{0, 1\}^\lambda$.

5.4 Challenge Injection

As we are attempting to solve an OW-CPA challenge, we are given a public key G and a ciphertext c^* and asked to find the (x^*, e^*) such that $c^* = x^*G + e^*$.

To simulate a challenge, we will first select a uniformly random index $j \xleftarrow{\$} \{1, \dots, P\}$. Then, we will select a uniformly random seed $s \in \{0, 1\}^k$. We will run the encapsulation algorithm ParQ.Enc on the seed s , except that we will *not* query ErrGen($s||j$) to generate e_j , and thus not generate x_j and c_j . Thus we will have $c_1, \dots, c_{j-1}, c_{j+1}, \dots, c_P$ and K .

To finish the challenge encapsulation, we will select a uniformly random bit $b \in \{0, 1\}$. If $b = 0$, we will send K , and if $b = 1$ we will send a uniformly random $K' \in \{0, 1\}^\lambda$. We will send $C = (c_1, \dots, c_{j-1}, c^*, c_{j+1}, \dots, c_P)$ as the encapsulation.

OW Challenge Solution Extraction. We need to show that the adversary’s advantage in solving the IND-CCA2 challenge corresponds to an extractor’s ability to solve the OW-CPA challenge. Note that the only way for an adversary to distinguish the correct key from an incorrect one is by querying the s associated with each c_i to the KDF oracle. Without having done this, the adversary has no information on K and so she has no advantage in distinguishing a proper K from a random one. Therefore, the adversary’s advantage in distinguishing corresponds exactly to their ability to query (and thus find) s .

First, we show that the adversary’s probability of querying s to KDF without having queried an e_i for one of the c_i ’s to PRF (along with i) is negligibly small.

Without having queried some e_i to PRF, the plaintext values x_1, \dots, x_P provide no information on s . Recall that $s = x_i \oplus \text{PRF}(e_i||i)$. Then (x_1, \dots, x_P) can be thought of as P maskings of the same value s , with independent masking values. This contains no information about s , unless the adversary has queried at least one e_i to PRF.

Similarly, the values (e_1, \dots, e_P) provide no information about s , unless the adversary queries $s||i$ to ErrGen for some i . This happens with probability at most $q_{\text{ErrGen}}/2^k$. So as long as s is not queried to ErrGen and $e_i||i$ is not queried to PRF, then both (x_1, \dots, x_P) and (e_1, \dots, e_P) give no information about s , and so the encapsulation $C = (c_1, \dots, c_P)$ does not.

So we have shown that unless the adversary queries $e_i||i$ to PRF or $s||i$ to ErrGen (for any i), the encapsulation $C = (c_1, \dots, c_P)$ actually contains no information whatsoever about s . Therefore, the adversary can only query random seeds to KDF and so the probability that they query s to KDF is at most $q_{\text{KDF}}/2^k$.

If the adversary queries $e_i||i$ to PRF for any i , then they can easily find s and thus break the indistinguishability challenge. But (as we will establish next), since the adversary has not queried s to KDF or $s||i$ to ErrGen, the adversary has no ability to detect which ciphertext c_i corresponds to the OW challenge c^* . So if the adversary submits an e_i to PRF, with probability $1/P$, this e_i is in fact e^* , and we will solve the OW-CPA challenge.

Indistinguishability of Simulated Challenge. When the adversary is given a challenge encapsulation $C = (c_1, \dots, c_{j-1}, c^*, c_{j+1}, \dots, c_P)$, along with a possible key K , we need to ensure that they cannot tell that this is not a correctly formatted encapsulation. Other than replacing c_j with c^* , this is a correct encapsulation. All encapsulations come in the form of P uniform ciphertexts. However a correct encapsulation has the additional property that for each (x_i, e_i) associated with a c_i , $s = x_i \oplus \text{PRF}(e_i||i)$ is the same for all c_i , and $e_i = \text{ErrGen}(s||i)$.

Intuitively, we can see that the only way for an adversary to distinguish between a correctly formatted encapsulation, and one that is generated as in our simulation is by being able to find the (x_i, e_i) associated with at least one of the c_i , and then checking the other c_i through the PRF and ErrGen functions.

Formally, if s has not been queried to *kdf*, $s||i$ has not been queried to ErrGen for any i , and $e_i||i$ has not been queried to PRF for any i , then each x_i and e_i is indistinguishable from being independently and uniformly generated. As such,

the ciphertext is perfectly indistinguishable unless the adversary queries $e_i||i$ to PRF for some i . This event also corresponds to the adversary's ability in solving the IND challenge and is considered in the previous subsection.

5.5 Simulating Decapsulation Queries

When we receive a query for decapsulation $C = (c_1, \dots, c_P)$, we need to respond with the decapsulation K . Upon receiving the query, we lookup the ErrGen table for P queries of the form $s||1, s||2, \dots, s||P$ such that c_i is in the table for each $s||i$. If such a set of P queries is found, we respond with $\text{KDF}(s)$. Otherwise, we return the decryption failure symbol \perp .

We must establish that this simulation is indistinguishable from a real decapsulation oracle. To establish this, we need to show two things: that we do not respond with \perp when we should respond with a decapsulated key, and that we do not respond with a decapsulated key when we should respond with \perp . For the first point, we must ensure that any potential encapsulation query made by the adversary in any way *other* than by beginning with a seed s and generating each c_i according to $c_i = \text{QCMDPC.Enc}(s \oplus \text{PRF}(\text{ErrGen}(s||i)||i), \text{ErrGen}(s||i))$ only results in a ciphertext that would not return \perp with negligible probability.

As previously noted, any ciphertext and index pair (c, i) is associated with exactly one seed s induced by $s = x \oplus \text{PRF}(e||i)$, as there is at most one pair (x, e) associated with c . For the ciphertext to be valid (and thus for a decapsulation oracle to *not* output \perp), it must be the case that $\text{ErrGen}(s||i) = e$. So for a decapsulation query $C = (c_1, \dots, c_P)$, for a correct decapsulation oracle to not return \perp , each c_i must be associated with the same seed s , and for each i , $e_i = \text{ErrGen}(s||i)$.

When an adversary submits a decapsulation query, if it is not the case that a single s has been queried P times to ErrGen in the form $s||1, s||2, \dots, s||P$, then there are two possibilities. Either for at least one c_i , no query has been made of the form $s' || i$ that results in c_i , or such a query has been made but the s' is different from one other s .

In the latter case, our simulation would return \perp , and indeed this is consistent with what an actual decapsulation oracle would return, as each c_i is not associated with the same seed, which the decapsulation algorithm can always detect.

In the first case, where $s||i$ has not been queried to generate c_i , our simulation will return \perp . This is usually consistent with what a correct decapsulation oracle will return. The only case an inconsistency would arise is if, when $\text{ErrGen}(s||i)$ is later queried, $\text{ErrGen}(s||i) = e_i$, despite it not having been queried at the time that the decapsulation query is made. As ErrGen is a random oracle, this only happens with probability at most $1/\#\mathbb{E}$.

Showing that we do not respond with a decapsulation when we should respond with \perp corresponds to the fact that we will never have a decoding failure. In a real decapsulation oracle, if a decoding error were to occur for each c_i , then we would be forced to respond with \perp . But in our simulated version, we would respond with the correct decapsulation, as we would have seen it from the

random oracle. However, because of the parallelization, we can see that any s will result in errors that will give a total decapsulation failure (i.e. the decoding procedure fail for all e_1, e_2, \dots, e_P) with probability ζ^P , where ζ is the decoding failure rate. Given this, we need to consider the probability that an adversary queries an encapsulation C that should result in a decapsulation failure. We should note that it should be hard to identify error vectors which will result in decoding failures (or else an adversary may not need to launch the GJS attack at all), but as we have no proof of this, we assume an adversary can perfectly distinguish which error vectors will result in decoding failures.

A fraction ζ^P of seeds will result in an encapsulation that cannot be decapsulated. So in q_{ErrGen} queries to the random oracle, the probability that the adversary is able to find such a seed is less than $q_{\text{ErrGen}}\zeta^P$. We assume that P is chosen so that this quantity is negligible (we discuss this further in Sect. 5.7).

5.6 Combining

We let Game 0 (or G_0) refer to the original IND-CCA2 game. We let Game 1 (G_1) refer to the simulated IND-CCA2 game, where the challenge and decapsulation oracle are simulated.

To simplify our calculation, we also define three events that can occur in the process of either Game 0 or Game 1.

- Event 1 (or E_1) refers to the event that the adversary \mathcal{A} queries s to the KDF oracle.
- Event 2 (or E_2) refers to the event of the adversary \mathcal{A} querying one of the $e_i||i$ (from the challenge encapsulation) to PRF prior to querying s to KDF or $s||i$ to ErrGen.
- Event 3 (or E_3) is the event that the adversary \mathcal{A} breaks the distinguishability of the simulated decapsulation oracle. Specifically, that they query an $s||i$ to ErrGen such that $\text{ErrGen}(s||i)$ will result in a decoding failure for each i , or that they submit a ciphertext to the decapsulation oracle without querying the associated s to construct it, and that when s is later queried, it does result in the proper error vector, and that they do this prior to Event 1 or 2.

Then, according to the discussion in Sects. 5.4 and 5.5, we perform the following calculation:

$$\begin{aligned} \frac{1}{2} + \epsilon &= \Pr_{G_0}[\mathcal{A} \text{ wins}] \\ &\leq \Pr_{G_0}[\mathcal{A} \text{ wins} | \neg E_1] + \Pr_{G_0}[E_1] \leq \frac{1}{2} + \Pr_{G_0}[E_1]. \end{aligned} \quad (2)$$

This tells us that $\epsilon \leq \Pr_{G_0}[E_1]$. Next, we consider $\Pr_{G_0}[E_1]$:

$$\epsilon \leq \Pr_{G_0}[E_1] \leq \Pr_{G_0}[E_2] + \Pr_{G_0}[E_1 | \neg E_2] \leq \Pr_{G_0}[E_2] + \frac{q_{\text{KDF}} + q_{\text{ErrGen}}}{2^k}. \quad (3)$$

Next, we relate $\Pr_{G_0}[E2]$ to $\Pr_{G_1}[E2]$. This is done simply by noting that

$$\Pr_{G_0}[E2] \leq \Pr_{G_0}[E2|\neg E3] + \Pr_{G_0}[E3], \quad (4)$$

and that

$$\Pr_{G_0}[E2|\neg E3] = \Pr_{G_1}[E2|\neg E3], \quad \Pr_{G_0}[E3] = \Pr_{G_1}[E3]. \quad (5)$$

Then finally, noting that our ability to solve the OW-CPA challenge corresponds to $1/P$ times $\Pr_{G_1}[E2 \wedge \neg E3]$, we get that

$$\begin{aligned} \Pr[\text{We win OW-CPA game}] &\geq \frac{1}{P} \Pr_{G_1}[E2 \wedge \neg E3] \\ &= \frac{1}{P} \Pr_{G_0}[\neg E3] \Pr_{G_0}[E2|\neg E3] \geq \frac{1}{P} \Pr_{G_0}[\neg E3] \left(\Pr_{G_0}[E2] - \Pr_{G_0}[E3] \right), \end{aligned} \quad (6)$$

and so

$$\Pr[\text{We win OW-CPA game}] \geq \frac{\gamma}{P} (\epsilon - \delta), \quad (7)$$

where

$$\delta = \frac{q_{\text{ParQ.Dec}}}{\#\mathbb{E}} + \frac{q_{\text{KDF}} + q_{\text{ErrGen}}}{2^k} + q_{\text{ErrGen}} \zeta^P \quad (8)$$

and

$$\gamma = 1 - \frac{q_{\text{ParQ.Dec}}}{\#\mathbb{E}} - q_{\text{ErrGen}} \zeta^P. \quad (9)$$

5.7 Comparison

In this section we compare aspects of ParQ's efficiency and security with other code-based KEMs, many of which have been submitted to NIST's Post-Quantum Cryptography project [1]. We restrict ourselves to code-based systems for direct comparison. Comparing code-based systems to other post-quantum systems has been done elsewhere in the literature, for example in [5]. All comparisons are done considering parameters that have been proposed for 128 bits of post-quantum security, or NIST's security level 5 (AES 256) (see Table 1).

While we do not have specific data on the speed of ParQ as it compares to other systems, one can expect that, because it requires P encapsulations and the decapsulation must be constant time to avoid side-channel timing attacks, the time to encapsulate and decapsulate likely increases by a factor of roughly P as opposed to a construction like CAKE.

Here we have selected the parameter P to be 12. This reflects the fact that it reduces the decapsulation error rate to be on the order of 2^{-252} , presumably hard for even a fully quantum adversary to find a seed that results in a total decapsulation failure (even if the adversary is perfectly able to tell which errors will result in decoding failures, which is presumably hard without the secret key). One could choose P to be much lower, on the order of 2 or 3. While it appears that the GJS attack would be mitigated by these low values of P , (increasing the

Table 1. Length in bytes of keys and encapsulations for code-based KEMs. Using 8192128 for classic McEliece

Scheme	Public key	Secret key	Encapsulation	Static key use
CAKE [5]	8193	8193	8225	✗
BIKE-1 [2]	8188	8188	8188	✗
BIKE-2 [2]	4094	8188	4094	✗
BIKE-3 [2]	9033	9033	9033	✗
Classic McEliece [6]	1 357 824	14 080	240	✓
ParQ	4094	8193	98 313	✓

attack complexity to an estimated 2^{66} or 2^{87} queries respectively), decapsulation errors may still occur in the normal lifetime of a key, meaning that the guarantees of the CCA2 security proof would not apply. While we have specified that P must be at least 2, note that P could be set to 1. This would cause the scheme to bear some resemblance to the CAKE scheme [5] or the BIKE-1 scheme [2]. However, this would cause the scheme to be vulnerable to the GJS attack, which is why these schemes currently insist on using the public key ephemeraly.

6 Conclusion and Future Work

We have explored and answered several fundamental questions that arose as a result of the powerful GJS reaction attack on QC-MDPC McEliece. We analyzed the origin of this leak: a bias on the distribution of the syndrome weight. This analysis allows a better understanding of the GJS attack and we deduce other side-channel attacks exploiting all decoding instances.

Our analysis provides quantitative bounds on the minimal number of samples needed to deduce relevant information (using Chernoff’s bound), which could be used to deduce better parameters to prevent attacks on the syndrome weight. Other side-channel attacks on different (noisier) parameters exploiting the same idea will be even more costly.

We also discussed how variations in the implemented decoding procedure can affect the attack. Lastly we have showed how decoding failures can be addressed at the protocol level by constructing a KEM that entirely defeats the GJS reaction attack for QC-MDPC, without altering the parameters of the system. We provided a proof of the CCA2 security of the KEM in the random oracle model. Notably, this proof considered the possibility of decoding failures, meaning that it should not be possible to attack the system by exploiting decoding failures.

The security of ParQ is proven in the random-oracle model. A complete and thorough analysis of post-quantum security would require a security reduction in the quantum random-oracle model [8]. Showing that ParQ (or a small modification of ParQ) is secure in this model would give greater post-quantum assurance.

MDPC codes are still a recent proposal. Even though they are close to the thoroughly studied LDPC codes, they seem to behave differently, in particular

as far as decoding is concerned [9]. It is very likely that the state of the art for decoding MDPC codes will evolve quickly, especially considering the NIST call for quantum safe primitives. Interestingly, it seems that more efficient decoders (*e.g.* those using variable threshold rules) are more prone to information leakage, and thus better decoders might not be safer. Evaluating new decoding algorithm, their failure rates and running time distribution with respect to this work could indicate whether and at what cost QC-MDPC codes could be used for PKEs as safely as for KEMs.

A Security Definitions and Games

These standard definitions, used in the security proof for ParQ, have been replicated from [15] for the sake of completeness.

The IND-CCA2 and OW-CPA games take place between two parties, the challenger \mathcal{C} , and the attacker or adversary, \mathcal{A} .

Game 1 (IND-CCA2 Challenge).

1. \mathcal{C} obtains $(pk, sk) \leftarrow \text{ParQ.KeyGen}(1^\lambda)$, and sends pk to \mathcal{A} . \mathcal{C} runs $\text{ParQ.Enc}(s)$ with a uniformly random s , obtaining K_0 , \mathcal{C} . \mathcal{C} then generates a uniformly random $K_1 \in \{0, 1\}^\lambda$, and a uniformly random bit $b \in \{0, 1\}$. \mathcal{C} then sends C and K_b to \mathcal{A} .
2. \mathcal{A} may freely send decapsulation queries C to \mathcal{C} . \mathcal{C} responds by sending $\text{ParQ.Dec}(C)$ to \mathcal{A} . The only exception is that \mathcal{A} may not send the challenge encapsulation C as a decapsulation query.
3. Eventually, \mathcal{A} must return a bit b' as a guess for the bit b . \mathcal{A} is said to have won the IND-CCA2 game if $b' = b$.

We write \mathcal{A} 's ability to win Game 1 as $1/2 + \epsilon$. We call ϵ the adversary's *advantage* in breaking IND-CCA2 security.

Game 2 (OW-CPA Challenge).

1. \mathcal{C} generates $(pk, sk) \leftarrow \text{QCMDPC.KeyGen}(1^\lambda)$. They select a uniformly random $x \xleftarrow{\$} \{0, 1\}^k$ and $e \xleftarrow{\$} \{0, 1\}^n$, with e having weight t . They then compute $c^* \leftarrow \text{QCMDPC.Enc}(pk, x, e)$ and sends c^* and pk to \mathcal{A} .
2. \mathcal{A} performs some computation on c^* and pk . Eventually they must produce an x' . \mathcal{A} is said to have won the OW-CPA game if $x' = x$.

B Choosing the Bit-Flipping Thresholds

In standard literature, rules for threshold computation are heuristic and are not available for all parameter sets. To convince that our experiments were fair we describe the rules we used for fixed and variable threshold. We denote $d = w/2$ the column weight.

Monitoring Strategy: For a given set of parameters, we run the bit-flipping algorithm on many random instances and we choose at each iteration the threshold which minimizes the error weight at the end of all flips¹. This is possible in a simulation because we know the initial error pattern and we can monitor its evolution. We will refer to this as the “monitoring strategy” and use it as a tool to define the thresholds.

Fixed Thresholds: For a given set of parameters, we run a simulation using the monitoring strategy and we keep track of the threshold values used at the first iteration. The maximum of those values is kept as the fixed threshold, say b_0 , for the first iteration. We run a second simulation, for which the first threshold is fixed to b_0 and the monitoring strategy is used for the following iterations. We keep track of the threshold values used at the second iteration. The maximum of those values is kept as the fixed threshold, say b_1 , for the second iteration. We repeat this until we reach the maximal expected number of iterations.

Variable thresholds: For a given set of parameters, the goal here is to establish a rule $b_i(\sigma)$, $i \geq 0$, giving the i -th iteration threshold as a function of the syndrome weight σ . Assuming all b_ℓ for $\ell < i$ are known, we run a simulation using the functions b_0, \dots, b_{i-1} for the first i iterations and using the monitoring strategy after that. We keep track of the pairs (σ, b) of syndrome weights and threshold values used at the i -th iteration. For each syndrome weight σ , we define $f_i(\sigma)$ as the average of all thresholds observed. Next, using the least square method, we find the quadratic² function $g_i(\sigma)$ which best approximates all the $(\sigma, f_i(\sigma))$ where each $(\sigma, f_i(\sigma))$ is weighted by the number of occurrences of the syndrome weight σ . The threshold function for the i -th iteration will be $\lceil g_i(\sigma) \rceil$. We add the condition that b_i is increasing with σ and we get $b_i : \sigma \rightarrow \max(b_i^{\min}, \lceil g_i(\sigma) \rceil)$ where b_i^{\min} is the minimal value of $\lceil g_i(\sigma) \rceil$ over the observed range for σ , and is never smaller than $d/2$.

Results and Comments. We give below the threshold rules we used for our simulations deduced from the above-mentioned process. Note that we do not claim, nor observed, that those rules are giving any kind of improvement in speed or failure rate.

Fixed Thresholds. For 80-bit security parameters, $(k, w, t) = (4801, 90, 84)$, we have $(b_i)_{i \geq 0} = (30, 28, 26, 25, 23, \dots)$. The dots meaning that the last value is repeated as much as necessary. We remark that, for the same parameters, QcBits [13] uses thresholds that are exactly one unit lower for the first 4 iterations. This probably reflects the fact that our strategy is rather conservative.

For 128-bit security, $(k, w, t) = (10163, 142, 134)$, we get $(b_i)_{i \geq 0} = (46, 43, 41, 40, 39, 37, 36, \dots)$. Finally for 256-bit security, $(k, w, t) = (32771, 274, 264)$ we obtain $(b_i)_{i \geq 0} = (83, 80, 77, 74, 72, \dots)$.

¹ In case of a tie, we choose the smallest threshold, but never smaller than $d/2$.

² We use the linear approximation unless the quadratic approximation gives different values of $b_i(\sigma) = \lceil g_i(\sigma) \rceil$ for σ in the observed range.

Variable Thresholds.

$$(k, w, t) = (4801, 90, 84) \Rightarrow \begin{cases} b_0(\sigma) = \lceil 11.1 + 0.00919 \sigma \rceil \\ b_1(\sigma) = \max(24, \lceil 38.7 - 0.0242\sigma + 1.004 \cdot 10^{-5} \sigma^2 \rceil) \\ b_i(\sigma) = \max(24, \lceil 34.9 - 0.0195\sigma + 0.836 \cdot 10^{-5} \sigma^2 \rceil), i \geq 2, \end{cases}$$

$$(k, w, t) = (10163, 142, 134) \Rightarrow \begin{cases} b_0(\sigma) = \lceil 15.5 + 0.00665 \sigma \rceil \\ b_1(\sigma) = \lceil 51.7 - 0.0128 \sigma + 0.257 \cdot 10^{-5} \sigma^2 \rceil \\ b_i(\sigma) = \max(37, \lceil 40.1 - 0.00395 \sigma + 9.50 \cdot 10^{-7} \sigma^2 \rceil), i \geq 2 \end{cases}$$

$$(k, w, t) = (32771, 274, 264) \Rightarrow \begin{cases} b_0(\sigma) = \lceil 22.9 + 0.00402 \sigma \rceil \\ b_1(\sigma) = \lceil 18.2 + 0.00431 \sigma \rceil \\ b_2(\sigma) = \max(71, \lceil 315.8 - 0.0422 \sigma + 0.182 \cdot 10^{-5} \sigma^2 \rceil) \\ b_i(\sigma) = \max(69, \lceil 62.5 + 0.000648 \sigma \rceil), i \geq 3. \end{cases}$$

References

1. NIST post-quantum cryptography project, round 1 submissions (2017). <https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions>
2. Aragon, N., Barreto, P.S.L.M., Battaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Güneysu, T., Melchor, C.A., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.P., Zémor, G.: BIKE—bit flipping key encapsulation (2017). <http://bikesuite.org>
3. Augot, D., Batina, L., Bernstein, D.J., Bos, J., Buchmann, J., Castryck, W., Dunkelmann, O., Güneysu, T., Gueron, S., Hülsing, A., Lange, T., Mohamed, M.S.E., Rechberger, C., Schwabe, P., Sendrier, N., Vercauteren, F., Yang, B.Y.: Initial recommendations of long-term secure post-quantum systems (2015). <http://pqcrypto.eu.org/docs/initial-recommendations.pdf>
4. Baldi, M., Bodrato, M., Chiaraluce, F.: A new analysis of the McEliece cryptosystem based on QC-LDPC codes. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 246–262. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85855-3_17
5. Barreto, P.S.L.M., Gueron, S., Güneysu, T., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.P.: CAKE: code-based algorithm for key encapsulation. Cryptology ePrint Archive, Report 2017/757 (2017)
6. Bernstein, D.J., Chou, T., Lange, T., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Wang, W.: Classic McEliece (2017). <https://classic.mceliece.org>
7. Bernstein, D.J., Chou, T., Schwabe, P.: McBits: fast constant-time code-based cryptography. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 250–272. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_15
8. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3
9. Chaulet, J.: Étude de cryptosystèmes à clé publique basés sur les codes MDPC quasi-cycliques. Ph.D. thesis, Université Pierre et Marie Curie-Paris VI (2017)

10. Chaulet, J., Sendrier, N.: Worst case QC-MDPC decoder for McEliece cryptosystem. In: IEEE International Symposium on Information Theory, (ISIT 2016), pp. 1366–1370 (2016)
11. Chen, C., Eisenbarth, T., von Maurich, I., Steinwandt, R.: Differential power analysis of a McEliece cryptosystem. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) ACNS 2015. LNCS, vol. 9092, pp. 538–556. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28166-7_26
12. Chen, C., Eisenbarth, T., von Maurich, I., Steinwandt, R.: Horizontal and vertical side channel analysis of a McEliece cryptosystem. *IEEE Trans. Inf. Forensics Secur.* **11**(6), 1093–1105 (2016)
13. Chou, T.: QcBits: constant-time small-key code-based cryptography. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 280–300. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_14
14. Deneuville, J.-C., Gaborit, P., Zémor, G.: Ouroboros: a simple, secure and efficient key exchange protocol based on coding theory. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 18–34. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_2
15. Dent, A.W.: A designer’s guide to KEMs. In: Paterson, K.G. (ed.) *Cryptography and Coding 2003*. LNCS, vol. 2898, pp. 133–151. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40974-8_12
16. Döttling, N., Dowsley, R., Müller-Quade, J., Nascimento, A.C.A.: A CCA2 secure variant of the McEliece cryptosystem. *IEEE Trans. Inf. Theory* **58**(10), 6672–6680 (2012)
17. Fabšič, T., Hromada, V., Stankovski, P., Zajac, P., Guo, Q., Johansson, T.: A reaction attack on the QC-LDPC McEliece cryptosystem. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 51–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_4
18. Gaborit, P.: Shorter keys for code based cryptography. In: *Proceedings of WCC*, pp. 81–90 (2005)
19. Gallager, R.G.: Low-density parity-check codes. Ph.D. thesis, Massachusetts Institute of Technology (1963)
20. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016 Part I. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_29
21. Habib, M., McDiarmid, C., Ramirez-Alfonsin, J., Reed, B.: Probabilistic methods for algorithmic discrete mathematics, vol. 16. Springer Science & Business Media, Heidelberg (2013). <https://doi.org/10.1007/978-3-662-12788-9>
22. Heyse, S., von Maurich, I., Güneysu, T.: Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 273–292. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_16
23. Kobara, K., Imai, H.: Semantically secure McEliece public-key cryptosystems - conversions for McEliece PKC -. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 19–35. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_2
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. von Maurich, I., Güneysu, T.: Towards side-channel resistant implementations of QC-MDPC McEliece encryption on constrained devices. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 266–282. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_16
26. von Maurich, I., Heberle, L., Güneysu, T.: IND-CCA secure hybrid encryption from QC-MDPC niederreiter. In: Takagi, T. (ed.) PQCrypto 2016. LNCS, vol. 9606, pp. 1–17. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29360-8_1
27. von Maurich, I., Oder, T., Güneysu, T.: Implementing QC-MDPC McEliece encryption. *ACM Trans. Embed. Comput. Syst. (TECS)* **14**(3), 44:1–44:27 (2015)
28. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. *Deep Space Netw. Prog. Rep.* **44**, 114–116 (1978)
29. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: new McEliece variants from moderate density parity-check codes. In: 2013 IEEE International Symposium on Information Theory, pp. 2069–2073 (2013)
30. Monico, C., Rosenthal, J., Shokrollahi, A.: Using low density parity check codes in the McEliece cryptosystem. In: IEEE International Symposium on Information Theory - ISIT 2000, p. 215. IEEE (2000)
31. Niederreiter, H.: Knapsack type of cryptosystems and algebraic coding theory **15**, 19–34 (1986)
32. Rosen, A., Segev, G.: Chosen-ciphertext security via correlated products. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 419–436. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_25
33. Strenzke, F.: A timing attack against the secret permutation in the McEliece PKC. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 95–107. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12929-2_8
34. Strenzke, F.: Timing attacks against the syndrome inversion in code-based cryptosystems. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 217–230. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_15
35. Strenzke, F., Tews, E., Molter, H.G., Overbeck, R., Shoufan, A.: Side channels in the McEliece PKC. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 216–229. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88403-3_15
36. Yoshida, Y., Morozov, K., Tanaka, K.: Ouroboros: a simple, secure and efficient key exchange protocol based on coding theory. In: PQCrypto 2017. LNCS, vol. 10346, pp. 35–50. Springer (2017)



FPGA-Based Niederreiter Cryptosystem Using Binary Goppa Codes

Wen Wang^{1(✉)}, Jakub Szefer^{1(✉)}, and Ruben Niederhagen^{2(✉)}

¹ Yale University, New Haven, CT, USA
{wen.wang.wv349, jakub.szefer}@yale.edu

² Fraunhofer SIT, Darmstadt, Germany
ruben@polycephaly.org

Abstract. This paper presents an FPGA implementation of the Niederreiter cryptosystem using binary Goppa codes, including modules for encryption, decryption, and key generation. We improve over previous implementations in terms of efficiency (time-area product and raw performance) and security level. Our implementation is constant time in order to protect against timing side-channel analysis. The design is fully parameterized, using code-generation scripts, in order to support a wide range of parameter choices for security, including binary field size, the degree of the Goppa polynomial, and the code length. The parameterized design allows us to choose design parameters for time-area trade-offs in order to support a wide variety of applications ranging from smart cards to server accelerators. For parameters that are considered to provide “128-bit post-quantum security”, our time-optimized implementation requires 966,400 cycles for the generation of both public and private portions of a key and 14,291 cycles to decrypt a ciphertext. The time-optimized design uses only 121,806 ALMs (52% of the available logic) and 961 RAM blocks (38% of the available memory), and results in a design that runs at about 250 MHz on a medium-size Stratix V FPGA.

Keywords: Post-Quantum Cryptography · Code-based cryptography
Niederreiter cryptosystem · FPGA · Hardware implementation

1 Introduction

Arguably today’s most wide-spread asymmetric cryptographic algorithms are the Rivest-Shamir-Adleman (RSA) cryptosystem, Diffie-Hellman key exchange (DH), and a variety of primitives from the field of Elliptic-Curve Cryptography (ECC), e.g., ECDSA, EdDSA, ECDH, etc. These cryptosystems are based on the hardness of the integer-factorization problem and the discrete-logarithm problem. Using today’s computing systems, no efficient algorithms for solving

Permanent ID of this document: 939f29123f6853e858d367a6a143be76.

Date: 2018.01.24.

these problems are known. However, the picture changes drastically if quantum computers are taken into account. In the 1990s, Shor proposed algorithms that can solve both the integer-factorization problem and the discrete-logarithm problem in polynomial time on a quantum computer [23, 24]. In order to provide alternatives to the threatened schemes, the field of Post-Quantum Cryptography (PQC) emerged in the 2000s and has received increased attention recently, most noticeably due to a standardization process for PQC schemes started by NIST in 2017 [7].

Currently, there are five categories of mathematical problems that are under investigation for PQC: code-based systems, lattice-based systems, hash-based systems, systems based on multivariate polynomial equations, and systems based on supersingular isogenies of elliptic curves [4, 22]. Each of these categories has advantages and disadvantages. They vary in the performance measures (sizes of public and private keys, sizes of ciphertext and key-exchange messages, computational cost, etc.) and in maturity: some schemes (e.g., some code-based schemes and hash-based signature schemes) are considered well-understood and there is a general agreement on the required security parameters while other schemes are more recent and the exact security that they provide is yet under investigation.

Conservative and well-understood choices for code-based cryptography are the McEliece cryptosystem [18] and its dual variant by Niederreiter [19] using binary Goppa codes. In this paper, we focus on the Niederreiter cryptosystem. This cryptosystem has relatively large public keys of up to 1 MB for roughly 256-bit classical security (corresponding to “128-bit post-quantum security” meaning that a quantum computer needs to perform at least 2^{128} “operations” using the best known attacks) using parameters proposed in [2]. There are more efficient PQC schemes than Niederreiter with binary Goppa codes. However, some of these schemes exhibit weaknesses that restrict their application to certain use-cases (e.g., Niederreiter with QC-MDPC codes instead of binary Goppa codes is affected by decoding errors [13] which restricts their use to ephemeral applications without long-term usage of keys) while how to choose security parameters for some schemes is challenging (e.g., for some lattice-based schemes that have a security reduction, parameters need to be chosen either based on best-known attacks or based on the non-tight security reduction, which results in a dilemma of choosing either more efficient or more reliable parameters [1]).

The large public keys of the Niederreiter cryptosystem using binary Goppa codes make it particularly troublesome for use in embedded systems (due to strong restrictions on resource usage) and in server scenarios (given a large number of simultaneous connections). In both cases, hardware acceleration can help to improve the performance—either by providing a low-area, power efficient crypto core in the embedded scenario or by providing a large, latency or throughput optimized crypto accelerator for the server scenario. Therefore, we describe and evaluate an FPGA implementation of this cryptosystem. Our FPGA implementation can be tuned in regard to performance and resource usage for either low-resource usage in embedded systems or high performance as accelerator for servers. Furthermore, we provide a generic implementation that can be used for

different performance parameters. This enables us to synthesize our design for the above mentioned 256-bit security parameters and also smaller parameter sets for comparison with prior art. For a given set of parameters, i.e. security level, the design can be further configured to trade-off performance and area, by changing widths of data paths, memories, and other parameters inside the design, without affecting the security level. All of the parameters can be configured for key generation, encryption, and decryption.

Inspired by the confidence in the code-based cryptosystems, there are a few hardware implementations of different variants of these cryptosystems, e.g., [14, 17, 26]. Most of the work only focuses on the encryption and decryption parts of the cryptosystem due to the complexity of the key generation module. Moreover, none of the prior designs are fully configurable as ours nor do they support the recommended “128-bit post-quantum security” level. We are aware of only one publication [26] that provides the design of a full McEliece cryptosystem including key generation, encryption and decryption modules. However, their design only provides a 103-bit classical security level, which does not meet the currently recommended security level for defending against quantum computers. More importantly, the design in [26] is not constant-time and has potential security flaws. For example, within their key generation part, they generate non-uniform permutations, and within the decryption part, they implement a non-constant-time decoding algorithm. Note that our work focuses on a design that can defend against timing side-channel attacks due to its constant-time implementation. However, other types of side-channel attacks are out of scope of this work. A detailed comparison with related work is presented in Section 5.

Contributions. This paper presents the first “128-bit post-quantum secure”, constant-time, efficient, and tunable FPGA-based implementation of the Niederreiter cryptosystem using binary Goppa codes. The contributions are:

- full cryptosystem with tunable parameters, which uses code-generation to generate vendor-neutral Verilog HDL code,
- new hardware implementation of merge sort for obtaining uniformly distributed permutations,
- new optimization of the Gao-Mateer additive FFT for polynomial evaluation,
- hardware implementation of a constant-time Berlekamp-Massey decoding algorithm, and
- design testing using Sage reference code, iVerilog simulation, and output from real FPGA runs.

2 Niederreiter Cryptosystem

The first public-key encryption scheme based on coding theory was proposed in 1978 by McEliece [18], known as the McEliece public-key cryptosystem. In 1986, Niederreiter proposed a variant of the McEliece cryptosystem that uses a parity check matrix for encryption instead of a generator matrix as used by

McEliece. Furthermore, Niederreiter proposed to use Reed-Solomon codes, which were later shown to be insecure [27]. However, the Niederreiter cryptosystem using binary Goppa codes remains secure and the Niederreiter cryptosystem has been shown to be equivalent (using corresponding security parameters) to the McEliece cryptosystem [15].

The private key of the Niederreiter cryptosystem is a binary Goppa code \mathcal{G} that is able to correct up to t errors. It consists of two parts: a generator, which is a monic irreducible polynomial $g(x)$ of degree t over $\text{GF}(2^m)$, and a support, which is a random sequence of n distinct elements from $\text{GF}(2^m)$. The public key is a binary parity check matrix $H \in \text{GF}(2)^{mt \times n}$, which is uniquely defined by the binary Goppa code. To reduce the size of the public key, the matrix H of size $mt \times n$ can be compressed to a matrix $K \in \text{GF}(2)^{mt \times k}$ of size $mt \times (n - mt)$ with $k = (n - mt)$ by computing its systematic form. This is often called “modern Niederreiter” and can also be used for the McEliece cryptosystem. For encryption, the sender encodes the message as a weight- t error vector e of length n . Then e is multiplied with the public parity check matrix H and the resulting syndrome is sent to the receiver as the ciphertext c . For decryption, the receiver uses the secret support and the generator to decrypt the ciphertext in polynomial time using an efficient syndrome decoding algorithm of \mathcal{G} . If neither the support nor the generator is known, it is computationally hard to decrypt the ciphertext, given only the public key H . The Niederreiter cryptosystem has performance advantages over the McEliece system if it is used as a key-encapsulation scheme, where a symmetric key is derived from the weight- t error vector e . The Niederreiter cryptosystem with properly chosen parameters is believed to be secure against attacks using quantum computers.

Security Parameters. The PQCRYPTO project [21] gives “initial recommendations” for several PQC schemes. For McEliece and Niederreiter using binary Goppa codes, they recommend to use a binary field of size $m = 13$, adding $t = 119$ errors, code length $n = 6960$, and code rank $k = n - mt = 6960 - 13 \cdot 119 = 5413$ for “128-bit post-quantum security” [2]. More precisely, these parameters give a classical security level of 266-bit (slightly overshooting 256-bit security); they were chosen to provide maximum security for a public key size of at most 1 MB [6]. We use these recommended parameters as primary target for our implementation. However, since our design is fully parameterized, we can synthesize our implementation for any meaningful choice of m , t , n , and k for comparison with prior art (see Section 5).

2.1 Algorithms

There are three main operations within the Niederreiter cryptosystem: key generation, encryption and decryption. Key generation is the most expensive operation; it is described in Algorithm 1. The implementation of the key generator has been described in detail in [28]. To generate a random sequence of distinct field

Algorithm 1. Key-generation algorithm for the Niederreiter cryptosystem.**Input** : System parameters: m , t , and n .**Output**: Private key $(g(x), (\alpha_0, \alpha_1, \dots, \alpha_{n-1}))$ and public key K .

- 1 Choose a random sequence $(\alpha_0, \alpha_1, \dots, \alpha_{n-1})$ of n distinct elements in $\text{GF}(2^m)$ as support.
- 2 Choose a random polynomial $g(x)$ as generator such that $g(\alpha) \neq 0$ for all $\alpha \in (\alpha_0, \dots, \alpha_{n-1})$.
- 3 Compute the $t \times n$ parity check matrix

$$H = \begin{bmatrix} 1/g(\alpha_0) & 1/g(\alpha_1) & \cdots & 1/g(\alpha_{n-1}) \\ \alpha_0/g(\alpha_0) & \alpha_1/g(\alpha_1) & \cdots & \alpha_{n-1}/g(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{t-1}/g(\alpha_0) & \alpha_1^{t-1}/g(\alpha_1) & \cdots & \alpha_{n-1}^{t-1}/g(\alpha_{n-1}) \end{bmatrix}.$$

- 4 Transform H to a $mt \times n$ binary parity check matrix H' by replacing each entry with a column of m bits.
- 5 Transform H' into its systematic form $[\mathbb{I}_{mt} | K]$.
- 6 Return the private key $(g(x), (\alpha_0, \alpha_1, \dots, \alpha_{n-1}))$ and the public key K .

Algorithm 2. Encryption algorithm for the Niederreiter cryptosystem.**Input** : Plaintext e , public key K .**Output**: Ciphertext c .

- 1 Compute $c = [\mathbb{I}_{mt} | K] \times e$.
- 2 Return the ciphertext c .

Algorithm 3. Decryption algorithm for the Niederreiter cryptosystem.**Input** : Ciphertext c , secret key $(g(x), (\alpha_0, \alpha_1, \dots, \alpha_{n-1}))$.**Output**: Plaintext e .

- 1 Compute the double-size $2t \times n$ parity check matrix

$$H^{(2)} = \begin{bmatrix} 1/g^2(\alpha_0) & 1/g^2(\alpha_1) & \cdots & 1/g^2(\alpha_{n-1}) \\ \alpha_0/g^2(\alpha_0) & \alpha_1/g^2(\alpha_1) & \cdots & \alpha_{n-1}/g^2(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1}/g^2(\alpha_0) & \alpha_1^{2t-1}/g^2(\alpha_1) & \cdots & \alpha_{n-1}^{2t-1}/g^2(\alpha_{n-1}) \end{bmatrix}.$$

- 2 Transform $H^{(2)}$ to a $2mt \times n$ binary parity check matrix $H'^{(2)}$ by replacing each entry with a column of m bits.
- 3 Compute the double-size syndrome: $S^{(2)} = H'^{(2)} \times (c|0)$.
- 4 Compute the error-locator polynomial $\sigma(x)$ by use of the decoding algorithm given $S^{(2)}$.
- 5 Evaluate the error-locator polynomial $\sigma(x)$ at $(\alpha_0, \alpha_1, \dots, \alpha_{n-1})$ and determine the plaintext bit values.
- 6 Return the plaintext e .

elements, [28] presents a low-cost Fisher-Yates shuffle module which generates a uniform permutation. However, the runtime of the permutation module in [28] depends on the generated secret random numbers. This non-constant-time design of the permutation module might have vulnerabilities which enable timing side-channel analysis. In our work, we present a merge sort module, which generates a uniform permutation within constant time, as described in Section 3.1.

Within the Niederreiter cryptosystem, the ciphertext is defined as a syndrome, which is the product between the parity check matrix and the plaintext. As shown in Algorithm 2, the encryption operation is very simple and maps to the multiplication between the extended public key $[\mathbb{I}_{mt}|K]$ and the plaintext e . In our work, we only focus on the core functionalities of the Niederreiter cryptosystem, therefore we assume that the input plaintext e is an n -bit error message of weight t .

As shown in Algorithm 3, the decryption operation starts from extracting the error locator polynomial out of the ciphertext using a decoding algorithm. We use the Berlekamp-Massey's (BM) algorithm in our design and develop a dedicated BM module for decoding, as described in Section 3.2. One problem within BM-decoding is that it can only recover $\frac{t}{2}$ errors. To solve this issue, we use the trick proposed by Nicolas Sendrier [14]. We first compute the double-size parity check matrix $H^{(2)}$ corresponding to $g^2(x)$, then we append $(n - mt)$ zeros to c . Based on the fact that e and $(c|0)$ belong to the same coset given $H^{(2)} \times (c|0) = H \times e$, computing the new double-size syndrome $S^{(2)}$ enables the BM algorithm to recover t errors. Once the error locator polynomial is computed, it is evaluated at the secret random sequence $(\alpha_0, \alpha_1, \dots, \alpha_{n-1})$, and finally the plaintext e is recovered.

2.2 Structure of the Paper

The following sections introduce the building blocks for our cryptosystem in a bottom-up fashion. Details of the $\text{GF}(2^m)$ finite field arithmetic and of the higher-level $\text{GF}(2^m)[x]/f$ polynomial arithmetic can be found in [28]. Leveraging the arithmetic operations are modules that are used in key generation, encryption, and decryption. For key generation, the description of the Gaussian systemization and additive FFT module has been provided in [28] and in this paper we will focus on the introduction of the new merge sort module and the optimization of the additive FFT module, as described in Section 3. For encryption, a simple matrix-vector multiplication is needed. For decryption, additive FFT is used as well, and a new Berlekamp-Massey decoding module is introduced and described in Section 3. Then we describe how these modules work together to obtain an efficient design for the full cryptosystem in Section 4. Validation of the design using Sage, iVerilog, and Stratix V FPGAs is presented in Section 5 together with a discussion and comparison with related work.

Algorithm 4. Fisher-Yates shuffle

Output: Shuffled array A **Initialize:** $A = \{0, 1, \dots, n - 1\}$

- 1 **for** i from $n - 1$ downto 0 **do**
 - 2 Generate j uniformly from range $[0, i)$
 - 3 Swap $A[i]$ and $A[j]$
-

Algorithm 5. Merge sort

Input: Random list A , of length 2^k **Output:** Sorted list A

- 1 Split A into 2^k sublists.
 - 2 **for** i from 0 to $k - 1$ **do**
 - 3 Merge adjacent sublists.
-

3 Modules

The main building blocks within our Niederreiter cryptosystem (as shown in Figure 2) are: two Gaussian systemizers for matrix systemization over $\text{GF}(2^m)$ and $\text{GF}(2)$ respectively, Gao-Mateer additive FFT for polynomial evaluations, a merge-sort module for generating uniformly distributed permutations, and a Berlekamp-Massey module for decoding. The Gaussian systemizer and the original version of additive FFT have been described in detail in [28]. We will focus on the merge-sort module, the Berlekamp-Massey module and our optimizations for the additive-FFT module in this section.

3.1 Random Permutation

An important step in the key-generation process is to compute a random permutation of selected field elements, which is part of the private key and therefore must be kept secret. In [28], the random permutation was computed by performing Fisher-Yates shuffle [11] on the ordered list $(0, 1, \dots, 2^m - 1)$. Algorithm 4 shows the operation of the Fisher-Yates shuffle. This algorithm computes a permutation efficiently and requires only a small amount of computational logic. As shown in Algorithm 4, in each iteration step i (in decrementing order), this module generates a random integer $0 \leq j < i$ (Algorithm 4, line 2), and then swaps the data in array position i and j . In [28], a PRNG is used, which keeps generating random numbers until the output is in the required range. Therefore, this implementation of Fisher-Yates shuffle produces a non-biased permutation (under the condition that the PRNG has no bias) but it is not constant-time because different seeds for the PRNG will lead to different cycle counts for the Fisher-Yates shuffle. This causes a potential risk of timing side-channel attacks, which is hard to eliminate even if a larger PRNG is used.

To fully eliminate potential timing attacks using the Fisher-Yates shuffle approach from [28], in this work, we implemented a constant-time sorting module for permutation based on the merge-sort algorithm. Sorting a random list can be regarded as the reverse operation of a permutation: Sorting a randomly permuted list can be seen as applying swapping operations on the elements until a sorted list is achieved. Applying the same swapping operations in reverse order to a sorted list results in a randomly permuted list. Therefore, given a constant-time sort algorithm, a constant-time algorithm for generating a random permutation can easily be derived.

Merge Sort. Merge sort is a comparison-based sorting algorithm which produces a stable sort. Algorithm 5 shows the merge sort algorithm. For example, a given random list $A = (92, 34, 18, 78, 91, 65, 80, 99)$ can be sorted by using merge sort within three steps: Initially, list A is divided into eight sublists $(92), (34), (18), (78), (91), (65), (80),$ and (99) with granularity of one. Since there is only one element in each sublist, these sublists are sorted. In the first step, all the adjacent sublists are merged and sorted, into four sublists $(34, 92), (18, 78), (65, 91),$ and $(80, 99)$ of size two. Merging of two sorted lists is simple: Iteratively, first elements of the lists are compared and the smaller one is removed from its list and appended to the merged list, until both lists are empty. In the second step, these lists are merged into two sublists $(18, 34, 78, 92)$ and $(65, 80, 91, 99)$ of size four. Finally, these two sublists are merged to the final sorted list $A_{\text{sorted}} = (18, 34, 65, 78, 80, 91, 92, 99)$.

In general, to sort a random list of n elements, merge sort needs $\log_2(n)$ iterations, where each step involves $O(n)$ comparison-based merging operations. Therefore, merge sort has an asymptotic complexity of $O(n \log_2(n))$.

Random Permutation. As mentioned above, sorting a random list can be regarded as the reverse operation of permutation. When given a random list A , before the merge sort process begins, we attach an index to each element in the list. Each element then has two parts: value and index, where the value is used for comparison-based sorting, and the index labels the original position of the element in list A . For the above example, to achieve a permutation for list $P = (0, 1, \dots, 7)$, we first attach an index to each of the elements in A , which gives us a new list $A' = ((92, 0), (34, 1), (18, 2), (78, 3), (91, 4), (65, 5), (80, 6), (99, 7))$. Then the merge sort process begins, which merges elements based on their value part, while the index part remains unchanged. Finally, we get $A'_{\text{sorted}} = ((18, 2), (34, 1), (65, 5), (78, 3), (80, 6), (91, 4), (92, 0), (99, 7))$. By extracting the index part of the final result, we get a random permutation of P , which is $(2, 1, 5, 3, 6, 4, 0, 7)$. In general, to compute a random permutation, we generate 2^m random numbers and append each of them with an index. The sorting result of these random numbers will uniquely determine the permutation.

In case there is a collision among the random values, the resulting permutation might be slightly biased. Therefore, the bit-width of the randomly generated numbers needs to be selected carefully to reduce the collision rate and thusly the

Design	Algorithm	Const.	Cycles	Logic	Time \times Area	Mem. Reg.	Fmax
[28]	FY-shuffle	\times	23,635	149	$3.52 \cdot 10^6$	7 111	334 MHz
Our	merge-sort	\checkmark	147,505	448	$6.61 \cdot 10^7$	46 615	365 MHz

Table 1. Performance of computing a permutation on $2^{13} = 8192$ elements with $m = 13$ and $b = 32$; Const. = Constant Time.

bias. If the width of the random numbers is b , then the probability that there are one or more collisions in 2^m randomly generated numbers is $1 - \prod_{i=1}^{2^m-1} \frac{(2^b-i)}{2^b}$ due to the birthday paradox. Therefore, for a given m , the collision rate can be reduced by using a larger b . However, increasing b also increases the required logic and memory. Both m and b are parameters which can be chosen at compile time in our implementation. The value for b can easily be chosen to fit to the required m . For the parameters $m = 13$ and $b = 32$ the collision rate is 0.0078. We further reduce the collision rate and thus the bias within merge sort by incorporating the following trick in our design at low logic cost: In case the two random to-be-merged values are equal, we do a conditional swap based on the least significant bit of the random value. Since the least significant bit of the random value is random, this trick will make sure that if some random numbers are generated twice, we can still get a non-biased permutation. There still is going to be a bias in the permutation if some random values appear more than two times. This case could be detected and the merge sort module could be restarted repeatedly until no bias occurs. However, the probability of this is very low (prob $\approx 2^{-27.58}$ according to [10]) for $m = 13$ and $b = 32$.

Fully Pipelined Hardware Implementation. We implemented a parameterized merge sort module using two dual-port memory blocks P and P' of depth 2^m and width $(b+m)$. First, a PRNG is used, which generates 2^m random b -bit strings, each cell of memory block P then gets initialized with one of the random b -bit strings concatenated with an m -bit index string (corresponding to the memory address in this case). Once the initialization of P finishes, the merge sort process starts. In our design, the merge sort algorithm is implemented in a pipelined way. The basic three operations in the merge-sort module are: read values from two sublists, compare the two values, and write down the smaller one to a new list. In our design, there are four pipeline stages: issue reads, fetch outputs from memory, compare the outputs, and write back to the other memory. We built separate logic for these four stages and time-multiplex these four stages by working on independent sublists in parallel whenever possible. By having the four-stage pipelines, we achieve a high-performance merge-sort design with a small logic overhead.

Table 1 shows a comparison between our new, constant time, sort-based permutation module with the non-constant time Fisher-Yates shuffle approach in [28]. Clearly, the constant-time permutation implementation requires more time, area, and particularly memory. Therefore, a trade-off needs to be made

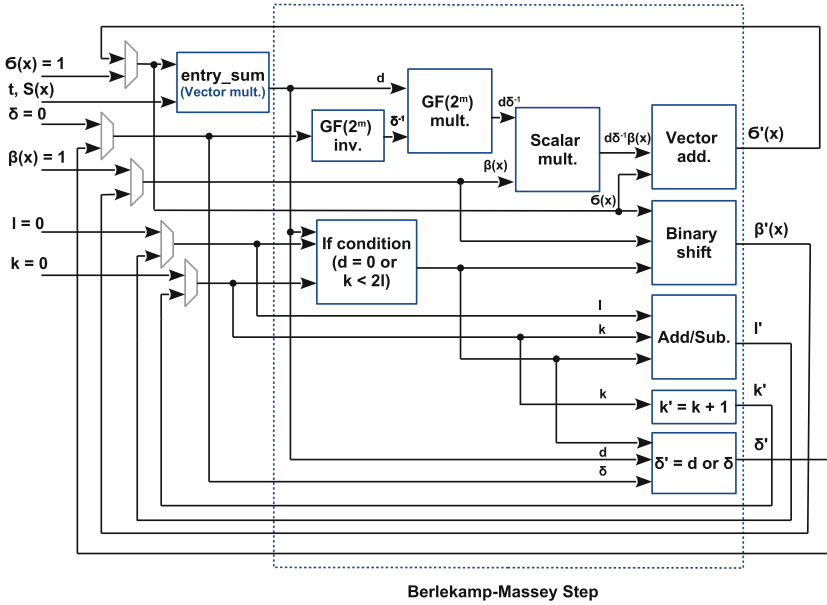


Figure 1. Dataflow diagram of the Berlekamp-Massey module.

between the need for increased security due to the constant-time implementation and resource utilization. In scenarios where timing side-channel protection is not needed, the cheaper Fisher-Yates shuffle version might be sufficient.

3.2 Berlekamp-Massey Algorithm

Finding a codeword at distance t from a vector v is the key step in the decryption operation. We apply a decoding algorithm to solve this problem. Among different algorithms, the Berlekamp-Massey (BM) algorithm [16] and Patterson’s algorithm [20] are the algorithms most commonly used. Patterson’s algorithm takes advantage of certain properties present in binary Goppa codes, and is able to correct up to t errors for binary Goppa codes with a designated minimum distance $d_{min} \geq 2t + 1$. On the other hand, general decoding algorithms like the BM algorithm can only correct $\frac{t}{2}$ errors by default, which can be increased to t errors using the trick proposed by Nicolas Sendrier [14]. However, the process of BM algorithm is quite simple compared to Patterson’s algorithm. More importantly, it is easier to protect the implementation of BM algorithm against timing attacks given the simplicity of the decryption steps. Consequently, we use BM algorithm in our decryption module.

Our implementation follows the Berlekamp iterative algorithm as described in [16]. The algorithm begins with initializing polynomials $\sigma(x) = 1 \in GF(2^m)[x]$, $\beta(x) = x \in GF(2^m)[x]$, integers $l = 0$ and $\delta = 1 \in GF(2^m)$. The input syndrome polynomial is denoted as $S(x) = \sum_{i=1}^{2t-1} S_i x^i \in GF(2^m)[x]$. Then

Algorithm 6. Berlekamp-Massey algorithm for decryption.

Input : Public security parameter t , syndrome polynomial $S(x)$.
Output: Error locator polynomial $\sigma(x)$.

- 1 **Initialize**: $\sigma(x) = 1$, $\beta(x) = x$, $l = 0$, $\delta = 1$.
- 2 **for** k from 0 to $2t - 1$ **do**
- 3 $d = \sum_{i=0}^t \sigma_i S_{k-i}$
- 4 **if** $d = 0$ or $k < 2l$:
- 5 $\{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\beta(x), l, \delta\}$.
- 6 **else**:
- 7 $\{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\sigma(x), k - l + 1, d\}$.
- 8 **Return** the error locator polynomial $\sigma(x)$.

within each iteration step k ($0 \leq k \leq 2t - 1$), the variables $\{\sigma(x), \beta(x), l, \delta\}$ are conditionally updated using operations described in Algorithm 6. Note that updating polynomial $\beta(x)$ only involves multiplying a polynomial by x , which can be easily mapped to a binary shifting operation on its coefficients in hardware. Updating integer l and field element δ only involves subtraction/addition operations, and these operations can also be easily implemented in hardware. Therefore the bottleneck of the algorithm lies in computing d and updating $\sigma(x)$.

Hardware Implementation. The first step within each iteration is to calculate d (Algorithm 6, line 3). We built an `entry_sum` module (as shown in Figure 1) for this computation, which maps to a vector-multiplication operation as described in [28]. We use two registers σ_{vec} and β_{vec} of $m \cdot (t + 1)$ bits to store the coefficients of polynomials $\sigma(x)$ and $\beta(x)$, where the constant terms σ_0 and β_0 are stored in the lowest m bits of the registers, σ_1 and β_1 are stored in the second lowest m bits, and so on. We also use a register S_{vec} of $m \cdot (t + 1)$ bits to store at most $(t + 1)$ coefficients of $S(x)$. This register is updated within each iteration, where S_k is stored in the least significant m bits of the register, S_{k-1} is stored in the second least significant m bits, and so on. The computation of d can then be regarded as an entry-wise vector multiplication between register σ_{vec} and register $S_{vec} = (0, 0, \dots, S_0, S_1, \dots, S_{k-1}, S_k)$ for all $0 \leq k \leq 2t - 1$. Register σ_{vec} is initialized as $(0, 0, \dots, 1)$ for the first iteration, and then gets updated with the new coefficients of $\sigma(x)$ for the next iteration. S_{vec} is initialized as all zeroes, and then constructed gradually by reading from a piece of memory which stores coefficient S_i of syndrome polynomial $S(x)$ at address i for $0 \leq i \leq 2t - 1$. Within the k -th iteration, a read request for address k of the memory is issued. Once the corresponding coefficient S_k is read out, it is inserted to the lowest m bits of S_{vec} . After the computation of d , we start updating variables $\{\sigma(x), \beta(x), l, \delta\}$. To update $\sigma(x)$, one field-element inversion, one field-element multiplication, one scalar multiplication as well as one vector subtraction are needed. At first, field element δ is inverted. As described in [28], the inversion of elements in $\text{GF}(2^m)$ can be implemented by use of a pre-computed lookup table. Each entry of the table can be read in one clock cycle. After reading

mul_{BM}	$\text{mul}_{\text{BM_step}}$	Cycles	Logic	Time \times Area	Mem. Reg.	Fmax
10	10	7379	6285	$4.64 \cdot 10^7$	7 13,089	364 MHz
20	20	4523	7052	$3.19 \cdot 10^7$	7 13,031	353 MHz
30	30	3571	7889	$2.82 \cdot 10^7$	7 12,956	361 MHz
40	40	3095	9047	$2.8 \cdot 10^7$	7 13,079	356 MHz
60	60	2619	11,400	$2.99 \cdot 10^7$	7 13,274	354 MHz

Table 2. Performance of the Berlekamp-Massey module for $m = 13$, $t = 119$, and $\deg(S(x)) = 237$.

out δ^{-1} , a field-element multiplication between d and δ^{-1} is performed, which makes use of the $\text{GF}(2^m)$ multiplication module as described in [28]. Once we get $d\delta^{-1}$, a scalar multiplication between field element $d\delta^{-1}$ and polynomial $\beta(x)$ starts, which can be mapped to an entry-wise vector multiplication between vector $(d\delta^{-1}, d\delta^{-1}, \dots, d\delta^{-1})$ and $(\beta_t, \beta_{t-1}, \dots, \beta_1, \beta_0)$. The last step for updating $\sigma(x)$ is to subtract $d\delta^{-1}\beta(x)$ from $\sigma(x)$. In a binary field $\text{GF}(2^m)$, subtraction and addition operations are equivalent. Therefore, the subtraction between $\sigma(x)$ and $d\delta^{-1}\beta(x)$ can simply be mapped to bit-wise **xor** operations between vector $(\sigma_t, \sigma_{t-1}, \dots, \sigma_1, \sigma_0)$ and vector $(d\delta^{-1}\beta_t, d\delta^{-1}\beta_{t-1}, \dots, d\delta^{-1}\beta_1, d\delta^{-1}\beta_0)$. Updating polynomial $\beta(x)$ is done by conditionally replacing its coefficient register β_{vec} with δ_{vec} , and then shift the resulting value leftwards by m bits. Updating integer l and field element δ only involves simple and cheap hardware operations.

The above iterations are repeated for a fixed number of $2t$ times, where t is the public security parameter. After $2t$ iterations, the final output is determined as the error locator polynomial $\sigma(x)$. It is easy to see that within each iteration, the sequence of instructions is fixed, as long as we make sure that the conditional updates of variables $\{\sigma(x), \beta(x), l, \delta\}$ are constant time (which is easy to achieve due to its fixed computational mapping in hardware), the run time of the whole design is fixed given the fixed iteration times. Therefore our BM implementation is fully protected against existing timing side-channel attacks, e.g., [3, 25].

We built a two-level design. The lower level is a **BM_step** module, which maps to one iteration, shown as “Berlekamp-Massey Step” in Figure 1. The higher-level BM module then iteratively applies **BM_step** and **entry_sum** modules. Table 2 shows performance for the BM module. A time-area trade-off can be achieved by adjusting the design parameters mul_{BM} and $\text{mul}_{\text{BM_step}}$, which are the number of multipliers used in the BM and **BM_step** modules. mul_{BM} and $\text{mul}_{\text{BM_step}}$ can be freely chosen as integers between 1 and $t + 1$.

3.3 Optimizations for Additive FFT

Evaluating a polynomial at multiple data points over $\text{GF}(2^m)$ is an essential step in both the key generation and the decryption processes. In key generation, an evaluation of the Goppa polynomial $g(x)$ is needed for computing the parity check matrix H , while for decryption, it is required by the computation of the double-size parity check matrix $H^{(2)}$ as well as the evaluation of the error

locator polynomial $\sigma(x)$. Therefore, having an efficient polynomial-evaluation module is very important for ensuring the performance of the overall design. We use a characteristic-2 additive FFT algorithm introduced in 2010 by Gao and Mateer [12], which was used for multipoint polynomial evaluation by Bernstein et al. in [5]. Additive FFT consists of two parts. First, radix conversion and twist is performed on the input polynomial. Given a polynomial $g(x)$ of 2^k coefficients, the recursive twist-then-radix-conversion process returns 2^k 1-coefficient polynomials. Then, these 1-coefficient polynomials are used to iteratively evaluate the input points by use of the reduction process.

We applied some modifications and improvements to both parts of the additive FFT design from [28]:

Optimizing Radix Conversion and Twisting. The radix-conversion step, which includes both radix conversion and twist, consists of several rounds that iteratively compute the new output coefficients of the converted input polynomial. The number of rounds is the base-2 logarithm of the degree of the input polynomial. In each round, new temporary coefficients are computed as the sum of some of the previous coefficients followed by a twist operation, i.e., a multiplication of each coefficient with a pre-computed constant to obtain a new basis for the respective round.

The radix-conversion module in [28] is using dedicated logic for each round for summing up the required coefficients, computing all coefficients within one cycle. Computing all coefficients with dedicated logic for each round requires a significant amount of area although radix conversion only requires a very small amount of cycles compared to the overall additive FFT process. Therefore, this results in a relatively high time-area product and a poor usage of resources.

We improve the area-time product at the cost of additional cycles and additional memory requirements by using the same logic block for different coefficients and rounds. An additional code-generation parameter is used to specify how many coefficients should be computed in parallel, which equals to the number of multipliers ($1 \leq \text{Mult.} \leq t + 1$) used in twist when mapping to hardware implementations. Each round then requires several cycles depending on the selected parameter. The computation of the new coefficients requires to sum up some of the previous coefficients. The logic therefore must be able to add up any selection of coefficients depending on the target coefficient. We are using round- and coefficient-dependent masks to define which coefficients to sum up in each specific case. These masks are stored in additional RAM modules.

Furthermore, in the design of [28], the length of the input polynomial is constrained to be a power of 2. For shorter polynomials, zero-coefficients need to be added, which brings quite some logic overhead especially on some extreme cases. For example, for a polynomial of 129 coefficients ($t = 128$), a size-256 radix conversion module will be needed. Instead, our improved design eliminates this constraint and allows an arbitrary input length with low overhead and therefore is able to further reduce cycle count and area requirements.

Design	Coeffs.	Mult.	Cycles	Logic	Time \times Area	Reg. Mem.	Fmax
Our	120	2	385	1893	$7.3 \cdot 10^5$	3541	6 305 MHz
Our	120	4	205	2679	$5.5 \cdot 10^5$	3622	10 273 MHz
[28]	128	4	211	5702	$1.2 \cdot 10^6$	7752	0 407 MHz
Our	120	8	115	4302	$4.9 \cdot 10^5$	3633	17 279 MHz
[28]	128	8	115	5916	$6.8 \cdot 10^5$	7717	0 400 MHz

Table 3. Performance of our radix-conversion module compared to [28] for $\text{GF}(2^{13})$.

Table 3 shows the performance improvements of the current radix-conversion module compared to the design in [28]. The numbers for our new design are given for a polynomial of length 120. The design in [28] requires the next larger power of 2 as input length. Therefore, we give numbers for input length 128 for comparison. For a processing width of four coefficients (multipliers), our new implementation gives a substantial improvement in regard to the time-area product over the old implementation at the cost of a few memory blocks.

Parameterizing Reduction. In the previous design of the additive FFT in [28], the configuration of the reduction module is fixed and uniquely determined by the polynomial size and the binary field size. Before the actual computation begins, the data memory is initialized with the 2^k 1-coefficient polynomials from the output of the last radix-conversion round. The data memory D within the reduction module is configured as follows: The depth of the memory equals to 2^k , based on this, the width of the memory is determined as $m \times 2^{m-k}$ since in total $m \times 2^m$ memory bits are needed to store the evaluation results for all the elements in $\text{GF}(2^m)$. Each row of memory D is initialized with 2^{m-k} identical 1-coefficient polynomials. The other piece of memory within the reduction module is the constants memory C. It has the same configuration as the data memory and it stores all the elements for evaluation of different reduction rounds. Once the initialization of data memory and constants memory is finished, the actual computation starts, which consists of the same amount of rounds as needed in the radix conversion process. Within each round, two rows of values (f_0 and f_1) are read from the data memory and the corresponding evaluation points from the constants memory, processed, and then the results are written back to the data memory. Each round of the reduction takes 2^k cycles to finish. In total, the reduction process takes $k \times 2^k$ cycles plus overhead for memory initialization.

In our current design, we made the reduction module parameterized by introducing a flexible memory configuration. The width of memories D and C can be adjusted to achieve a trade-off between logic and cycles. The algorithmic pattern for reduction remains the same, while the computational pattern changes due to the flexible data reorganization within the memories. Instead of fixing the memory width as $m \times 2^{m-k}$, it can be configured as a wider memory of width $m \times 2^{m-k+i}$, $0 \leq i \leq k$. In this way, we can store multiple 1-coefficient

Mult.	Cycles	Logic	Time × Area	Mem. bits	Mem.	Reg.	Fmax
32	968	4707	$4.56 \cdot 10^6$	212,160	63	10,851	421 MHz
64	488	9814	$4.79 \cdot 10^6$	212,992	126	22,128	395 MHz

Table 4. Performance of our parameterized size-128 reduction module for $\text{GF}(2^{13})$.

Multipliers								
Design	Rad.	Red.	Cycles	Logic	Time × Area	Mem.	Reg.	Fmax
Our	4	32	1173	7344	$8.61 \cdot 10^6$	73	14,092	274 MHz
[28]	4	32	1179	10,430	$1.23 \cdot 10^7$	63	18,413	382 MHz
Our	8	64	603	13,950	$8.41 \cdot 10^6$	143	25,603	279 MHz
[28]	8	32	1083	10,710	$1.16 \cdot 10^7$	63	18,363	362 MHz

Table 5. Performance of our optimized additive-FFT module compared to [28] for $m = 13$, $\deg(g(x)) = 119$. Rad. and Red. are the number of multipliers used in radix conversion and twist (reduction) separately.

polynomials at one memory address. The organization of the constants memory needs to be adapted accordingly. Therefore, within each cycle, we can either fetch, do computation on, or write back more data and therefore finish the whole reduction process within much fewer cycles ($k \times 2^{k-i}$ plus overhead of few initialization cycles). However, the speedup of the running time is achieved at the price of increasing the logic overhead, e.g., each time the width of the memory doubles, the number of multipliers needed for computation also doubles.

Table 4 shows the performance of our parameterized reduction module. We can see that doubling the memory width halves the cycles needed for the reduction process, but at the same time approximately doubles the logic utilization. We can see that although the memory bits needed for reduction remain similar for different design configurations, the number of required memory blocks doubles in order to achieve the increased memory width. Users can easily achieve a trade-off between performance and logic by tuning the memory configurations within the reduction module.

Table 5 shows performance of the current optimized additive FFT module. By tuning the design parameters in the radix conversion and reduction parts, we are able to achieve a 28% smaller time-area product compared to [28] when Rad. = 4 and Red. = 64.

4 Key Generation, Encryption and Decryption

We designed the Niederreiter cryptosystem by using the main building blocks shown in Figure 2. Note that we are using two simple 64-bit Xorshift PRNGs in our design to enable deterministic testing. For real deployment, these PRNGs

must be replaced with a cryptographically secure random-number generator, e.g., [8]. We require at most b random bits per clock cycle per PRNG.

4.1 Key Generation

The overall design of our key-generation module is identical to the design in [28]. The dataflow diagram is shown in Figure 2a. However, we improve the security of private-key generation by substituting the Fisher-Yates Shuffle module with a merge-sort module in order to generate a uniform and random permutation in constant time (see Section 3.1). The generation of the public key is improved by several optimizations applied to the additive FFT module (see Section 3.3).

Table 6 shows a comparison of the performance of the old implementation in [28] with our new, improved implementation. Despite the higher cost for the constant-time permutation module, overall, we achieve an improvement in regard to area requirements and therefore to the time-area product at roughly the same frequency on the price of a higher memory demand. However, the overall memory increase is less than 10% which we believe is justified by the increased side-channel resistance due to the use of a constant-time permutation.

4.2 Encryption

Figure 2b shows the interface of the encryption module. The encryption module assumes that the public key K is fed in column by column. The matrix-vector multiplication $[\mathbb{I}_{mt}|K] \times e$ is mapped to serial `xor` operations. Once the `PK_column_valid` signal is high, indicating that a new public-key column (`PK_column`) is available at the input port, the module checks if the corresponding bit of plaintext e is 1 or 0. If the bit value is 1, then an `xor` operation between the current output register (initialized as 0) and the new public-key column is carried out. Otherwise, no operation is performed. After the `xor` operation between K and the last $(n - mt)$ bits of e is finished, we carry out one more `xor` operation between the output register and the first mt bits of e . Then the updated value of the output register will be sent out as the ciphertext c . Table 7 shows performance of the encryption module. The encryption module is able to handle one column of the public key in each cycle and therefore requires a fixed number of $(n - mt)$ cycles independent of the secret input vector e .

4.3 Decryption

Within the decryption module, as described in Figure 2c, first the evaluation of the Goppa polynomial $g(x)$ is carried out by use of the optimized additive FFT module, which was described in Section 3.3. In our implementation, instead of first computing the double-size parity-check matrix $H^{(2)}$ and then computing the double-size syndrome $S^{(2)}$, we combine these two steps together. The computation of $S^{(2)}$ can be mapped to serial conditional `xor` operations of the columns of $H^{(2)}$. Based on the observation that the last $(n - mt)$ bits of vector $(c|0)$ are all

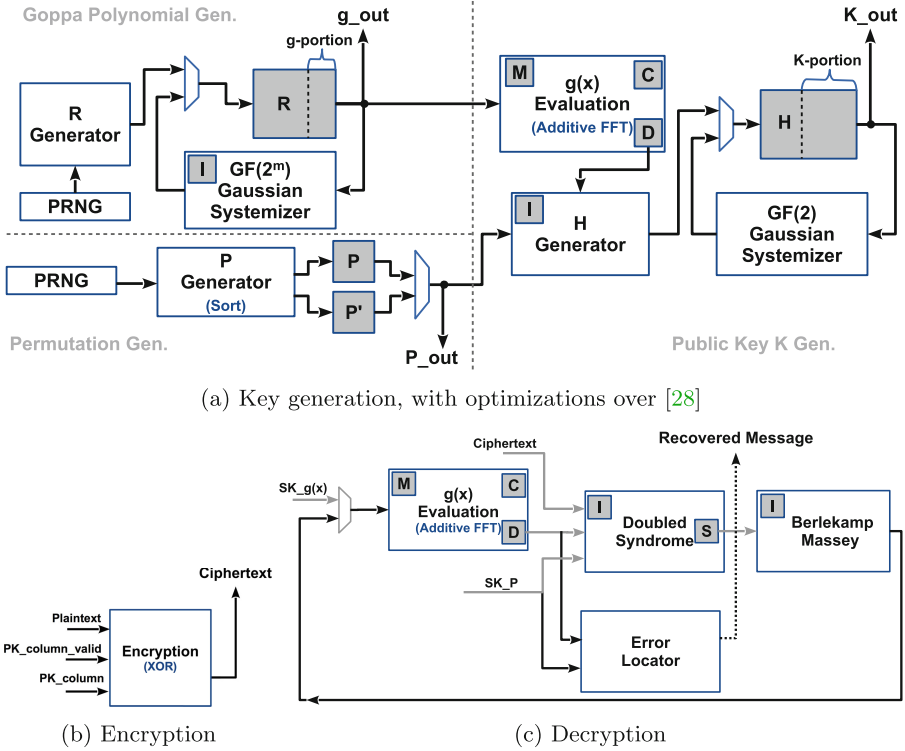


Figure 2. Dataflow diagrams of the three parts of the full cryptosystem: (a) key generation, (b) encryption, and (c) decryption. Dark gray boxes represent block memories, while white boxes represent major logic modules.

zero, the last $(n - mt)$ columns of $H^{(2)}$ do not need to be computed. Furthermore, the ciphertext c should be a uniformly random bit string. Therefore, for the first mt columns of $H^{(2)}$, roughly only half of the columns need to be computed. Finally, we selectively choose which columns of $H^{(2)}$ we need to compute based on the nonzero bits of the binary vector $(c|0)$. In total, approximately $m \times t^2$ field element multiplications are needed for computing the double-size syndrome. The computation of the corresponding columns of $H^{(2)}$ is performed in a similar column-block-wise method as described in [28]. The size B ($1 \leq B \leq \frac{mt}{2}$) of the column block is a design parameter that users can pick freely to achieve a trade-off between logic and cycles during computation. After the double-syndrome $S^{(2)}$ is computed, it is fed into the Berlekamp-Massey module described in Section 3.2 and the error-locator polynomial $\sigma(x)$ is determined as the output. Next, the error-locator polynomial $\sigma(x)$ is evaluated using the additive FFT module (see Section 3.3) at all the data points over $GF(2^m)$. Then, the message bits are determined by checking the data memory contents within the additive FFT module that correspond to the secret key-element set $(\alpha_0, \alpha_1, \dots, \alpha_{n-1})$. If the

Case	N_H	N_R	Cycles	Logic	Time \times Area Mem.	Fmax	Time
Prior work [28]							
Logic	40	1	11,121,220	29,711	$3.30 \cdot 10^{11}$	756 240 MHz	46.43 ms
Bal.	80	2	3,062,942	48,354	$1.48 \cdot 10^{11}$	764 248 MHz	12.37 ms
Time	160	4	896,052	101,508	$9.10 \cdot 10^{10}$	803 244 MHz	3.68 ms
Our work							
Logic	40	1	11,121,214	22,716	$2.53 \cdot 10^{11}$	819 237 MHz	46.83 ms
Bal.	80	2	3,062,936	39,122	$1.20 \cdot 10^{11}$	827 230 MHz	13.34 ms
Time	160	4	966,400	88,715	$8.57 \cdot 10^{10}$	873 251 MHz	3.85 ms

Table 6. Performance of the key-generation module for parameters $m = 13$, $t = 119$, and $n = 6960$. All the numbers in the table come from compilation reports of the Altera tool chain for Stratix V FPGAs.

m	t	n	Cycles	Logic	Time \times Area Mem. Reg.	Fmax
13	119	6960	5413	4276	$2.31 \cdot 10^7$	0 6977 448 MHz

Table 7. Performance for the encryption module.

Case	B	mul_{BM}	Cycles	Logic	Time \times Area Mem. Reg.	Fmax	Time
Area	10	10	34,492	19,377	$6.68 \cdot 10^8$	88 47,749 289 MHz	0.12 ms
Bal.	20	20	22,768	20,815	$4.74 \cdot 10^8$	88 48,050 290 MHz	0.08 ms
Time	40	40	17,055	23,901	$4.08 \cdot 10^8$	88 49,407 300 MHz	0.06 ms

Table 8. Performance for the decryption module for $m = 13$, $t = 119$ and $n = 6960$, $\text{mul}_{\text{BM_step}}$ is set to mul_{BM} .

corresponding evaluation result for α_i , $i = 0, 1, \dots, n - 1$ equals to zero, then the i -th bit of the plaintext is determined as 1, otherwise is determined as 0. After checking the evaluation results for all the elements in the set $(\alpha_0, \alpha_1, \dots, \alpha_{n-1})$, the plaintext is determined. Table 8 shows the performance of the decryption module with different design parameters. By tuning design parameters $\text{mul}_{\text{BM_step}}$, mul_{BM} , and B, a time-area trade-off can be made.

5 Testing, Evaluation, and Comparison

Our implementation of the Niederreiter cryptosystem is fully parameterized and can be synthesized for any choice of reasonable security parameters. However, the main target of our implementation is the 256-bit (classical) security level, which corresponds to a level at least “128-bit post-quantum security”. For testing, we used the parameters suggested in the PQCRYPTO recommendations [2]: $m = 13$, $t = 119$, $n = 6960$ and $k = 5413$ ($k = n - mt$).

Testing. To validate the FPGA implementation, in addition to simulations, we implemented a serial IO interface for communication between the host computer and the FPGA. The interface allows us to send data and simple commands from

Case	N_H	N_R	B	mul_{BM}	Logic	Mem.	Reg.	Fmax
Area	40	1	10	10	53,447 (23%)	907 (35%)	118,243	245 MHz
Bal.	80	2	20	20	70,478 (30%)	915 (36%)	146,648	251 MHz
Time	160	4	40	40	121,806 (52%)	961 (38%)	223,232	248 MHz

Table 9. Performance for the entire Niederreiter cryptosystem (i.e., key generation, encryption, and decryption) including the serial IO interface when synthesized for the Stratix V (5SGXEA7N) FPGA; $\text{mul}_{\text{BM_step}}$ is set to mul_{BM} .

the host to the FPGA and receive data, e.g., public and private key, ciphertext, and plaintext, from the FPGA. We verified the correct operation of our design by comparing the FPGA outputs with our Sage reference implementation (using the same PRNG and random seeds).

Evaluation. We synthesized our design using Altera Quartus 17.0 for these parameters on a Stratix V FPGA (5SGXEA7N). The results are given in Table 9, with included logic overhead of the IO interface. We provide numbers for three performance parameter sets, one for small area, one for small runtime, and one for balanced time and area. The parameters N_R and N_H control the size of the systolic array in the Gaussian systemizer modules, which are used for computing the private Goppa polynomial and the public key. Parameter B is the matrix-block size used for computing the syndrome. Parameter mul_{BM} determines the number of multipliers used in the high-level BM decoding module. The number of multipliers ($\text{mul}_{\text{BM_step}}$) used in the low-level `BM_step` module is set to mul_{BM} for the evaluation. The memory requirement varies slightly due the differences in the memory word size based on the design parameters. These design parameters can be freely chosen as long as the synthesized result fits on the target FPGA. For security parameter set $m = 13, t = 119, n = 6960$, our experiment shows that the largest design parameter set we can fit on Stratix V FPGA is: $N_R = 250, N_H = 6, \text{mul}_{\text{BM}} = 60, \text{mul}_{\text{BM_step}} = 60$, and $B = 60$.

Comparison. In the following, we compare our work with previous designs.

First, we compare it with a 103-bit classical security-level hardware-design described in [26]. This work is the only previously existing hardware implementation for the whole code-based cryptosystem, including a key generator, that we have found in literature. To compare with their work, we synthesized our design with the Xilinx tool-chain version 14.7 for a Virtex-5 XC5VLX110 FPGA. Note that the performance data of [26] in Table 10 includes a CCA2 conversion for encryption and decryption, which adds some overhead compared to our design. From Table 10, we can see that our design is much faster when comparing cycles and time, and also much cheaper in regard to area and memory consumption.

Second, we compare our work with a hardware design from [17], which presents the previously fastest decryption module for a McEliece cryptosystem. Therefore the comparison of our work with design [17] only focuses on the decryption part. We synthesized our decryption module with the parameters they used, which correspond to a 128-bit classical security level, for a Virtex-6

	Cycles			Logic	Freq. (MHz)	Mem.	Time (ms)		
	Gen.	Dec.	Enc.				Gen.	Dec.	Enc.
$m = 11, t = 50, n = 2048$, Virtex 5 LX110									
[28]	14,670,000	210,300	81,500	14,537	163	75	90.00	1.29	0.50
Our	1,503,927	5864	1498	6660	180	68	8.35	0.03	0.01
$m = 12, t = 66, n = 3307$, Virtex 6 LX240									
[17]	-	28,887	-	3307	162	15	-	0.18	-
Our	-	10,228	-	6571	267	23	-	0.04	-
$m = 13, t = 128, n = 8192$, Haswell vs. Stratix V									
[9]	1,236,054,840	343,344	289,152	-	4000	-	309.01	0.09	0.07
Our	1,173,750	17,140	6528	129,059	231	1126	5.08	0.07	0.07

Table 10. Comparison with related work. Logic is given in “Slices” for Xilinx Virtex FPGAs and in “ALMs” for Altera Stratix FPGAs.

XC6VLX240T FPGA. From Table 10, we can see that the time-area product of our decryption module is $10228 \cdot 6571 = 67,208,188$, which is 30% smaller than the time-area product of their design of $28887 \cdot 3307 = 95,529,309$ when comparing only the decryption module. Moreover, our design is able to achieve a much higher frequency and a smaller cycle counts compared to their design. Overall we are more than 4x faster than [17].

Finally, we also compare the performance of our hardware design with the to-date fastest CPU implementation of the Niederreiter cryptosystem [9]. In this case, we ran our implementation on our Altera Stratix V FPGA and compare it to a Haswell CPU running at 4 GHz. Our implementation competes very well with the CPU implementation, despite the over 10x slower clock of the FPGA.

6 Conclusion

This paper presented a complete hardware implementation of Niederreiter’s code-based cryptosystem based on binary Goppa codes, including key generation, encryption and decryption. The presented design can be configured with tunable parameters, and uses code-generation to generate vendor-neutral Verilog HDL code for any set of reasonable parameters. This work presented hardware implementations of an optimization of the Gao-Mateer additive FFT for polynomial evaluation, of merge sort used for obtaining uniformly distributed permutations, and of a constant-time Berlekamp-Massey algorithm.

Open-Source Code. The source code for this project is available under an open-source license at <http://caslab.csl.yale.edu/code/niederreiter/>.

Acknowledgments. This work was supported in part by United States’ National Science Foundation grant 1716541. We would like to acknowledge FPGA hardware donations from Altera (now part of Intel). We also want to thank Tung (Tony) Chou for his invaluable help. This paper has been greatly improved thanks to feedback from our shepherds Lajla Batina and Pedro Maat Costa Massolino and the anonymous reviewers.

References

1. Alkadri, N.A., Buchmann, J., Bansarkhani, R.E., Krämer, J.: A framework to select parameters for lattice-based cryptography. *Cryptology ePrint Archive*, Report 2017/615 (2017). <https://eprint.iacr.org/2017/615>
2. Augot, D., Batina, L., Bernstein, D.J., Bos, J., Buchmann, J., Castryck, W., Dunkelmann, O., Güneysu, T., Gueron, S., Hülsing, A., Lange, T., Mohamed, M.S.E., Rechberger, C., Schwabe, P., Sendrier, N., Vercauteren, F., Yang, B.Y.: Initial recommendations of long-term secure post-quantum systems. Technical report, PQCRYPTO ICT-645622 (2015). <https://pqcrypto.eu.org/docs/initial-recommendations.pdf>
3. Avanzi, R., Hoerder, S., Page, D., Tunstall, M.: Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. *JCEN* **1**(4), 271–281 (2011)
4. Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.): *Post-Quantum Cryptography*. Springer, Heidelberg (2009)
5. Bernstein, D.J., Chou, T., Schwabe, P.: McBits: fast constant-time code-based cryptography. In: Bertoni, G., Coron, J.-S. (eds.) *CHES 2013*. LNCS, vol. 8086, pp. 250–272. Springer, Heidelberg (2013)
6. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. In: Buchmann, J., Ding, J. (eds.) *PQCrypto 2008*. LNCS, vol. 5299, pp. 31–46. Springer, Heidelberg (2008)
7. Chen, L., Moody, D., Liu, Y.K.: NIST post-quantum cryptography standardization. <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/>
8. Cherkaoui, A., Fischer, V., Fesquet, L., Aubert, A.: A very high speed true random number generator with entropy assessment. In: Bertoni, G., Coron, J.-S. (eds.) *CHES 2013*. LNCS, vol. 8086, pp. 179–196. Springer, Heidelberg (2013)
9. Chou, T.: McBits revisited. In: Fischer, W., Homma, N. (eds.) *CHES 2017*. LNCS, vol. 10529, pp. 213–231. Springer, Cham (2017)
10. DasGupta, A.: The matching, birthday and the strong birthday problem: a contemporary review. *J. Stat. Plan. Inference* **130**(1), 377–389 (2005)
11. Fisher, R.A., Yates, F.: *Statistical Tables for Biological, Agricultural and Medical Research*. Oliver and Boyd, London (1948)
12. Gao, S., Mateer, T.: Additive fast Fourier transforms over finite fields. *IEEE Trans. Inf. Theory* **56**(12), 6265–6272 (2010)
13. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) *ASIACRYPT 2016*. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (2016)
14. Heyse, S., Güneysu, T.: Code-based cryptography on reconfigurable hardware: tweaking Niederreiter encryption for performance. *JCEN* **3**(1), 29–43 (2013)
15. Li, Y.X., Deng, R.H., Wang, X.M.: On the equivalence of McEliece’s and Niederreiter’s public-key cryptosystems. *IEEE Trans. Inf. Theory* **40**(1), 271–273 (1994)
16. Massey, J.: Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory* **15**(1), 122–127 (1969)
17. Massolino, P.M.C., Barreto, P.S.L.M., Ruggiero, W.V.: Optimized and scalable co-processor for McEliece with binary Goppa codes. *ACM Trans. Embed. Comput. Syst.* **14**(3), 45 (2015)
18. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. *DSN Progr. Rep.* **42–44**, 114–116 (1978)

19. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. *Probl. Control Inf. Theory* **15**, 19–34 (1986)
20. Patterson, N.: The algebraic decoding of Goppa codes. *IEEE Trans. Inf. Theory* **21**(2), 203–207 (1975)
21. Post-quantum cryptography for long-term security PQCRYPTO ICT-645622. <https://pqcrypto.eu.org/>
22. Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. *Cryptography ePrint Archive, Report 2006/145* (2006)
23. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: *Foundations of Computer Science - FOCS 1994*, pp. 124–134. IEEE (1994)
24. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **41**(2), 303–332 (1999)
25. Shoufan, A., Strenzke, F., Molter, H.G., Stöttinger, M.: A timing attack against patterson algorithm in the McEliece PKC. In: Lee, D., Hong, S. (eds.) *ICISC 2009*. LNCS, vol. 5984, pp. 161–175. Springer, Heidelberg (2010)
26. Shoufan, A., Wink, T., Molter, G., Huss, S., Strenzke, F.: A novel processor architecture for McEliece cryptosystem and FPGA platforms. *IEEE Trans. Comput.* **59**(11), 1533–1546 (2010)
27. Sidelnikov, V.M., Shestakov, S.O.: On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discret. Math. Appl.* **2**(4), 439–444 (1992)
28. Wang, W., Szefer, J., Niederhagen, R.: FPGA-based key generator for the Niederreiter cryptosystem using binary Goppa codes. In: Fischer, W., Homma, N. (eds.) *CHES 2017*. LNCS, vol. 10529, pp. 253–274. Springer, Cham (2017)

Cryptanalysis



Attacks on the AJPS Mersenne-Based Cryptosystem

Koen de Boer^{1(✉)}, Léo Ducas¹, Stacey Jeffery^{1,2}, and Ronald de Wolf^{1,2,3}

¹ CWI, Amsterdam, The Netherlands
kboer@cwi.nl

² QuSoft, Amsterdam, The Netherlands

³ University of Amsterdam, Amsterdam, The Netherlands

Abstract. Aggarwal, Joux, Prakash and Santha recently introduced a new potentially quantum-safe public-key cryptosystem, and suggested that a brute-force attack is essentially optimal against it. They consider but then dismiss both Meet-in-the-Middle attacks and LLL-based attacks. Very soon after their paper appeared, Beunardeau et al. proposed a practical LLL-based technique that seemed to significantly reduce the security of the AJPS system. In this paper we do two things. First, we show that a Meet-in-the-Middle attack can also be made to work against the AJPS system, using locality-sensitive hashing to overcome the difficulty that Aggarwal et al. saw for such attacks. We also present a quantum version of this attack. Second, we give a more precise analysis of the attack of Beunardeau et al., confirming and refining their results.

1 Introduction

Aggarwal et al. [1] recently proposed a variant of the NTRU public-key encryption scheme [13]. This variant uses integers with sparse binary representation as a secret key, rather than polynomials with small coefficients. In particular, their cryptosystem is suspected to be resistant to quantum attacks.

Their system works as follows. Consider a *Mersenne* number $N = 2^n - 1$, with n prime. Then we can identify the ring $R = \mathbb{Z}/N\mathbb{Z}$ with the set of n -bit strings, where 1^n is identified with 0^n . To set up the keys of the cryptosystem, choose $f, g \in R$ of fixed Hamming weight $w = \lfloor \sqrt{n}/2 \rfloor$ uniformly at random, subject to g having a multiplicative inverse in R . Set the *public* key to $h := f/g$ (this corresponds to an n -bit string of arbitrary Hamming weight) and the *private* key to g . In the next section we describe how Aggarwal et al. use these keys for encryption and decryption.

The security of this system relies on the assumption that it is hard to solve the following *Mersenne Low Hamming Ratio Search Problem*: given $n, w \in \mathbb{N}$ and $h \in R$, find $f, g \in R$ of weight w such that $h = f/g$, assuming such f and g exist. A brute-force attack on this system would just try out all $\binom{n-1}{w-1}$ possible g 's of weight w that start with a 1 (the latter is without loss of generality) and

check whether hg has weight w . Aggarwal et al. [1] suggest that this brute-force attack is close to optimal. This would correspond to roughly $\lambda = \log \binom{n-1}{w-1} \approx \frac{1}{2} \cdot w \log n$ bits of security. On a quantum computer, this brute-force attack could be implemented using Grover’s quantum search algorithm [11] in time roughly $\sqrt{\binom{n-1}{w-1}}$, corresponding to roughly $\frac{1}{4} \cdot w \log n$ bits of security.

In particular, Aggarwal et al. [1] consider and then dismiss two possible lines of attack that could be better than brute force. First, they suggest that a combinatorial Meet-in-the-Middle attack would fail due to a problem of “approximate collisions”. Second, they argue that their variation makes an adaptation of known lattice attacks against NTRU ineffective. The latter claim was rapidly challenged, when a faster experimental attack using LLL reduction was found by Beunardeau et al. [6]. This attack exploits the low weight of f and g , and is able to find f, g using partition-search in the integer interval $\{0, \dots, 2^n - 1\}$. The authors argue that their attack reduces the bit security to about $\lambda \approx 2w$. Beunardeau et al. [6] warn that their attack is only practically feasible, and might not work with, for example, increasing parameters. They further expect that slightly changing the cryptosystem protects against this attack [6, Sect. 4].

1.1 This Work

In this work we revisit the security of the AJPS cryptosystem. We first propose a Meet-in-the-Middle attack that circumvents the issues raised by [1] and gives a polynomial speed-up over a brute-force attack. It runs in classical time $\tilde{O}\left(\sqrt{\binom{n-1}{w-1}}\right)$, and can be accelerated on a quantum computer to $\tilde{O}\left(\sqrt[3]{\binom{n-1}{w-1}}\right)$. Our analysis requires several minor heuristics, which we have confirmed experimentally. Secondly, we formally analyze the attack of Beunardeau et al. [6]. Our analysis suggest that the attack is slightly less efficient, asymptotically, than suggested in [6]. However, this small difference in complexity makes little difference in practice.

Meet-in-the-Middle attack. Aggarwal et al. [1, Sect. 5.1] described a failed attempt at a Meet-in-the-Middle (MITM) attack on their cryptosystem. It fails because the “collisions” in the “middle” are not exact, and they view this failure as evidence for the optimality of the brute-force attack. In contrast, we show how a MITM attack on their system can nonetheless be executed, using locality-sensitive hashing to overcome the issue of inexact, approximate collisions.

The idea is still, given public key $h \in \{0, 1\}^n$, to find an n -bit string $g \in R$ of weight $\leq w$, such that hg also has low weight. Split the n -bit string $g = g_1 \oplus g_2$ into an n -bit string g_1 with roughly αw 1s in the first αn bits and 0s elsewhere, and a g_2 with roughly $(1 - \alpha)w$ 1s in the last $(1 - \alpha)n$ bits and 0s elsewhere. Now $hg = hg_1 + hg_2$ having low weight corresponds to hg_1 and hg_2 being approximately equal (i.e., having low Hamming distance), so our goal becomes to find an “approximate collision” between the two sets $\{hg_1\}$ and $\{hg_2\}$. We can do this by first computing all elements of the first set, together with their hashes, and storing these in an appropriate data structure. After that

we search in the second set to find an approximate collision with the elements in the data structure (if such an approximate collision exists). This attack turns out to be substantially cheaper than a brute-force search over all g 's of weight w . In the classical case, setting the split at $\alpha = 1/2$, the runtime of the attack is roughly $\binom{n/2}{w/2} \approx \binom{n}{w}^{1/2}$, which corresponds to roughly $\frac{1}{4} \cdot w \log n$ bits of security. In the quantum case, setting $\alpha = 1/3$ yields an algorithm similar to [5], which has runtime roughly $\binom{n/3}{w/3} \approx \binom{n}{w}^{1/3}$, corresponding to $\frac{1}{6} \cdot w \log n$ bits of security.

A meet-in-the-middle attack on NTRU, which has a similar structure to the AJPS cryptosystem, is due to Odlyzko, and is described in [15]. The first example of a quantum meet-in-the-middle algorithm was the collision-finding algorithm of Brassard et al. [9]. Similar ideas were later used in a quantum algorithm for the subset sum problem [5], and a quantum attack on the NTRU cryptosystem [23], which have a similar structure to the algorithm presented here. One difference in our new algorithm is the use of Ambainis's variable-cost quantum search algorithm [3], described in Sect. 2.2.

To complement our theoretical analysis we also implemented this attack on a classical computer and ran a simulation for quantum computers. Our source code is available at <https://github.com/lducas/MiTM-Mersenne>.

Analysis of the lattice attack of Beunardeau et al. Although Beunardeau et al. [6] provide experimental evidence for the efficiency of their attack, they leave open the task of providing a theoretical analysis to support the correctness of their approach. This leaves some uncertainty for a concrete security estimate of the cryptosystem of Aggarwal et al. We attempt to fill this gap with a more in-depth analysis of their attack. We conclude that the cost of their attack is in fact of the form $(2 + \delta + o(1))^{2w}$ for some very small constant $\delta > 0$. Besides clarifying the heuristic asymptotic complexity of the attack of Beunardeau et al. [6], it also essentially confirms their practical claim that their attack reduces the security to roughly $2w$ bits. Hence it remains the best known attack on the AJPS cryptosystem (better than our MITM attack).

1.2 Impact

The impact of this work is mostly of a conceptual nature. Our Meet-in-the-Middle attack is a reminder that inexact collisions can sometimes be circumvented, depending on the metric at hand. While a similar near-collision MITM attack was well known against NTRU (attributed to Odlyzko in [15]), it was rather easy due to how close the near-collisions were. The setting of Aggarwal et al. is more demanding. Our work also shows another application of Nearest-Neighbor Search (NNS) techniques to cryptanalysis, which have already found important application to lattice problems [4, 17, 18].

Our analysis of the attack of Beunardeau et al. [6] also provides better confidence in the revised security estimate of the treated cryptosystem [1]. Moreover, we hope that it provides clear tools and heuristics to understand the behavior of LLL in more general scenarios.

Open questions. Our work highlights several interesting open questions. Concerning the cryptosystem of [1], an interesting idea would be to see whether the lattice attack and the MITM attack could be combined into an even faster attack, as was already done against NTRU by Howgrave-Graham [14]. At first sight, it seems that this approach would not lead to an exponential acceleration, yet it may make it possible to amortize the polynomial cost of each call to the LLL algorithm.

More generally, our work highlights the question of Nearest-Neighbor Search using quantum computers. This question was already approached in [17, 19], which considered generic application of Grover’s algorithm over classical NNS techniques. It seems an important question to determine whether less generic approaches could perform better.

1.3 Organization

The remainder of this paper is organized as follows. In Sect. 2, we give the necessary preliminaries, including a description of the AJPS cryptosystem of [1], and a description of a variant of the quantum search algorithm due to [3], for settings where the cost of checking if an element is marked varies. In Sect. 3, we present and analyze our classical Meet-in-the-Middle attack, and in Sect. 4, we present and analyze our quantum Meet-in-the-Middle attack. In Sect. 5, we present our formal analysis of the Beunardeau et al. attack [6].

2 Preliminaries

2.1 The AJPS Cryptosystem

In this section, we will describe the cryptosystem of Aggarwal et al. [1] (the AJPS cryptosystem). Let $N = 2^n - 1$, where n is a prime number¹, and let $R = \mathbb{Z}/N\mathbb{Z}$ be the integer ring modulo N . We define $w = \lfloor \sqrt{n}/2 \rfloor$ to be the upper bound on what we will consider “low weight”.

We will identify a number in R with its binary representation. In this way, we can represent the elements of R by the elements of \mathbb{F}_2^n , with 1^n and 0^n both representing $0 \in R$, and all other elements of R having unique representatives in \mathbb{F}_2^n . For nonzero $a \in R$, denote by $|a|$ the Hamming weight of the unique binary representation of a , and define $|0| = 0$. Similarly, denote by $\Delta(a, b)$ the Hamming distance between the binary representations of a and b (using the representation 0^n for $0 \in R$). Note that it is not necessarily the case that $|ab|$ and $|a| \cdot |b|$ are equal nor that $|a + b|$ and $|a| + |b|$ are equal. However, we have the following.

¹ Numbers of the form $N = 2^n - 1$ with $n \in \mathbb{N}$ are called *Mersenne numbers*. If, additionally, $N = 2^n - 1$ is prime, it is called a *Mersenne prime*. For the purposes of the AJPS cryptosystem, N doesn’t need to be prime, but n does.

Lemma 1 ([1]). *Let $a, b \in R$. Then*

- (i) $|a + b| \leq |a| + |b|$,
- (ii) $|ab| \leq |a| \cdot |b|$, and
- (iii) if $a \neq 0$, then $|-a| = n - |a|$.

Elements of the ring R have the special property that for any $i \in \{0, \dots, n-1\}$ and $a \in R$, the binary representation of $a \cdot 2^i \pmod N$ is just a cyclic shift of the binary representation of a by i .

We now describe the AJPS cryptosystem [1] with public parameter n .

Key Generation. Randomly choose two elements $f, g \in R$ of Hamming weight w , where g is invertible in R . Set $h = f/g$. The public key is h , and the secret key is g .

Encryption. To encrypt a bit s , pick random $p, q \in R$ of Hamming weight at most w . Output the ciphertext $c = (-1)^s(ph + q) \in R$.

Decryption. To decrypt c , compute $cg = (-1)^s(phg + qg) = (-1)^s(pf + qg)$. Since p, q, f and g all have Hamming weight $\leq w$, the n -bit string $pf + qg$ has Hamming weight $\leq 2w^2 < n/2$ by Lemma 1. Thus if $s = 0$, then $|cg| < n/2$. On the other hand, if $s = 1$, then $|-cg| < n/2$, so by Lemma 1, $|cg| > n - n/2 = n/2$. Thus, to decrypt c , output 0 if $|cg| < n/2$, and 1 otherwise.

To attack this cryptosystem, it suffices to solve the following problem:

Given $h \in R$, find $g \in R$ of Hamming weight w such that $|hg| = w$, assuming such a g exists.

Since multiplication by 2^i just shifts the binary representation of an element of R by i , if g is a solution to the above problem, then so is $2^i g$. Thus, if a solution exists, then a solution with the first bit set to 1 exists, and so we can restrict our attention to such solutions. Since a brute-force attack can find such a solution g in time $\binom{n-1}{w-1}$, to achieve security parameter λ , n and w must satisfy $\binom{n-1}{w-1} > 2^\lambda$ and $w < \sqrt{n}/2$. Our results, however, imply that a stronger condition is required to achieve λ -bit security.

2.2 Quantum Search with Variable Costs

In this section we will introduce the quantum search algorithm, originally due to Grover [11] and later generalized [7,8]. This algorithm searches a universe of size N for a particular *marked* item, given access to some procedure for checking if a given item is marked, using $O(\sqrt{N})$ calls to the checking procedure. We will also make use of an elegant variant of the quantum search algorithm due to Ambainis [3], that has better complexity when the cost of checking if a given item is marked varies by item.

Let U be some set of N objects, and let $\mathcal{C} : U \rightarrow \{0, 1\}$ be some procedure, called the *checking* procedure, that outputs 1 when given a marked item, and suppose the complexity of the procedure \mathcal{C} is C . Then there exists a quantum

algorithm with complexity $\tilde{O}(C\sqrt{N})$ that outputs $u \in U$ such that $\mathcal{C}(u) = 1$ with probability at least $2/3$, assuming such a u exists.

We may also consider the scenario in which the complexity of computing $\mathcal{C}(u)$ varies with u . Call this complexity $C(u)$. Using the previously mentioned standard quantum search algorithm, we can search for $u \in U$ such that $\mathcal{C}(u) = 1$ in $\tilde{O}(\max_{u \in U} C(u)\sqrt{N})$ steps. However, we can do better:

Theorem 1 (Quantum search with variable costs (Ambainis [3])). *Let $\mathcal{C} : U \rightarrow \{0, 1\}$ be any checking procedure. There exists a quantum algorithm that outputs $u \in U$ such that $\mathcal{C}(u) = 1$ with probability at least $2/3$ (assuming such a u exists), and has complexity*

$$\tilde{O}\left(\sqrt{\sum_{u \in U} C(u)^2}\right),$$

where $C(u)$ is the cost of computing $\mathcal{C}(u)$, and needn't be known in advance. We call this algorithm quantum search for $u \in U$ that satisfies \mathcal{C} .

In the case where $C(u) = C$ is constant, the algorithm from Theorem 1 has complexity $\tilde{O}(C\sqrt{N})$, as in the standard quantum search algorithm.

3 Classical Meet-in-the-Middle Attack

3.1 Introduction

The ‘‘Meet-in-the-Middle attack’’ (MITM attack) is a well-known generic cryptographic attack that can be deployed against a variety of cryptosystems, often achieving an improved time complexity in breaking the system, at the cost of greater space complexity. It may have originated in [12].

To illustrate this attack, we give an example in the context of the knapsack problem, which can be described as follows. Given numbers $h_1, \dots, h_n \in \mathbb{Z}$, find $g \in \mathbb{F}_2^n$ such that

$$\sum_{i=1}^n h_i g[i] = 0.$$

The MITMA idea is to split $\mathbb{F}_2^n = G_1 \oplus G_2$ into two equally-large subspaces of dimension $n/2$, where $G_1 = \{(g, 0^{\lceil n/2 \rceil}) : g \in \mathbb{F}_2^{\lfloor n/2 \rfloor}\}$ and $G_2 = \{(0^{\lfloor n/2 \rfloor}, g) : g \in \mathbb{F}_2^{\lceil n/2 \rceil}\}$. We calculate all numbers $H(g_1) = -\sum_i h_i g_1[i]$ for $g_1 \in G_1$ and store them in a database \mathcal{D} . This costs $2^{n/2}$ time and space, up to a $\text{poly}(n)$ factor.

Hereafter, we calculate $H(g_2) = \sum_i h_i g_2[i]$ for $g_2 \in G_2$, and check whether the element $-H(g_2)$ is somewhere in \mathcal{D} , using a single database lookup. If so, then we have found a $g_1 \in G_1$ such that $H(g_2) = -H(g_1)$. Then $g_1 + g_2 \in \mathbb{F}_2^n$ is a solution. This search costs about $2^{n/2}$ database lookups, and $2^{n/2} \cdot \text{poly}(n)$ time. This has much better time complexity than trying all combinations, which costs roughly 2^n time (but $\text{poly}(n)$ space).

Description of our MITM attack on the AJPS system. Given $h \in R$, we want to find $f, g \in R$, each of Hamming weight w , such that $h = f/g$ — or equivalently, such that $gh = f$. In other words, our task is, informally, to find $g \in \mathbb{F}_2^n$ of weight w such that $|gh|$ is small.

For $\alpha \in [0, 1]$ to be specified later, we define

$$G_1^{(\alpha)} = \{(g, 0^{\lceil(1-\alpha)n\rceil}) : g \in \mathbb{F}_2^{\lfloor\alpha n\rfloor}, |g| = \lfloor\alpha w\rfloor\}$$

and $G_2^{(\alpha)} = \{(0^{\lfloor\alpha n\rfloor}, g) : g \in \mathbb{F}_2^{\lceil(1-\alpha)n\rceil}, |g| = \lceil(1-\alpha)w\rceil\}.$

We note that while $G_1^{(\alpha)} \oplus G_2^{(\alpha)}$ does not include all $g \in \mathbb{F}_2^n$ such that $|g| = w$, restricting to this set is without loss of generality, since, if $g = g_1 + g_2$ is a solution, meaning that both g and gh have weight w , then for any $z \in \{0, \dots, n-1\}$, $2^z g$ is also a solution. This is because $2^z g$ just shifts the binary representation of g by z in a cyclic manner, so $2^z g$ and $2^z gh$ also have weight w . Thus, if there exists a solution, then there exists a solution in which g_1 and g_2 have weights $\lfloor\alpha w\rfloor$ and $\lceil(1-\alpha)w\rceil$, respectively.

The attack will begin by enumerating $(g_1, g_1 h)$ for all $g_1 \in G_1^{(\alpha)}$, after which we will search over $G_2^{(\alpha)}$ for some g_2 that is in collision with some $g_1 \in G_1^{(\alpha)}$, where, intuitively, we want to define g_1, g_2 to be in collision whenever the Hamming distance $\Delta(g_1 h, -g_2 h)$ is not much bigger than $2w$. The difficulty is that, given some value $-g_2 h$, while it would be easy to find a stored value of $g_1 h$ that is *equal* to $-g_2 h$, it is not immediately clear how to find such a stored value that is *close in Hamming distance* to $-g_2 h$.

Locality-sensitive hash functions. Our solution is to use a simple form of locality-sensitive hashing [16]. Intuitively, a locality-sensitive hash should take the same value, with high probability, on two elements that are *close* with respect to some desired distance. In our case, for $\mathcal{B} = \{i_1, \dots, i_B\} \subset [n]$ with $i_1 < \dots < i_B$, define $\mathcal{H}_{\mathcal{B}} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^{|\mathcal{B}|}$ by $\mathcal{H}_{\mathcal{B}}(s_1, \dots, s_n) = (s_{i_1}, \dots, s_{i_B})$. We will use the function family $\mathcal{F}_B = \{\mathcal{H}_{\mathcal{B}} : |\mathcal{B}| = B\}$ for some B to be specified later. This works for our purposes, because if two strings are close in Hamming distance, then on a random small subset \mathcal{B} of their bits, they are likely to agree.

Detailed description of algorithm. Our Meet-in-the-Middle attack proceeds as follows:

1. Choose a uniformly random $\mathcal{H} \in \mathcal{F}_B$.
2. Initialize an empty hash table \mathcal{D} , with 2^B (initially empty) linked lists, one for each element in the range of \mathcal{H} .
3. For each $g_1 \in G_1^{(\alpha)}$:
 - (a) Insert $(g_1, \mathcal{H}(g_1 h))$ into \mathcal{D} .
4. For each $g_2 \in G_2^{(\alpha)}$:
 - (a) Look up $\mathcal{H}(-g_2 h)$ in \mathcal{D} , and let L be the resulting list of values g_1 such that $\mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)$.
 - (b) For each g_1 in L :
 - i. If $|g_1 + g_2| = w$ and $|(g_1 + g_2)h| = w$, then output $g_1 + g_2$.

Analysis of algorithm. We first argue that our algorithm succeeds in finding a solution if one exists. The next lemma shows that if $-g_2h$ is uniformly random from R , and b is an arbitrary element of R with Hamming weight w , then (with high probability) $-g_2h$ and $g_1h = -g_2h + b$ do not differ in much more than $2w$ bits of their binary representations, i.e., $\Delta(-g_2h, g_1h)$ is not much larger than $2w$.

Lemma 2. *Let $a \in R = \mathbb{Z}/(2^n - 1)\mathbb{Z}$ be chosen uniformly at random, let $w \in \mathbb{N}$ and let $b \in R$ be any element such that $|b| = w$. Then, for every $s > 0$, we have:*

$$\mathbb{P}[\Delta(a, a + b) > 2w + s\sqrt{w}] \leq \exp\left(-\frac{s^2}{8 + 4s/\sqrt{w}}\right) + 2^{-n}. \tag{1}$$

Proof. We assume a to be chosen uniformly at random from \mathbb{F}_2^n , instead of from R . These two distributions \mathcal{P}, \mathcal{Q} on the set of n -bit strings only differ by 2^{-n} with respect to the total variation distance:

$$\begin{aligned} \frac{1}{2} \sum_{b \in \mathbb{F}_2^n} |\mathcal{P}[b] - \mathcal{Q}[b]| &= \frac{1}{2} \left(|\mathcal{P}[1^n] - \mathcal{Q}[1^n]| + \sum_{b \in \mathbb{F}_2^n \setminus \{1^n\}} |\mathcal{P}[b] - \mathcal{Q}[b]| \right) \\ &= \frac{1}{2} \left(\frac{1}{2^n} + (2^n - 1) \left(\frac{1}{2^n - 1} - \frac{1}{2^n} \right) \right) = 2^{-n}, \end{aligned}$$

which accounts for the 2^{-n} -term in the right-hand side of Eq. (1).

Given a, b , we define the carry element $c_n c_{n-1} \dots c_1 = c \in R$ by $c = (a + b) \oplus (a \oplus b)$. One can show that c equals the ‘carry vector’ that one puts above the sum of a and b when doing addition on a blackboard (see Table 1). Note that $a \oplus b \oplus c = a + b$, implying that $\Delta(a, a + b) = |b \oplus c| = |b| + |c| - 2|b \wedge c| = w + |c| - 2|b \wedge c|$.

Table 1. Having a nonzero bit in $b \wedge c$ leads necessarily to an extra carry.

c	0111	1110
a	0010	1111
b	0001	0001
$b \wedge c$	0001	0000

We now want to analyze the number of carry bits, i.e., the random variable $|c|$. The idea of the proof is that each 1-bit in b , combined with bits of a , will induce a sequence of carry-bits “to its left”. If there were no other 1-bits in b , then that induced number of carry-bits would be geometrically distributed with parameter $1/2$; we can think of this as the number of 1s that precede the first 0 in a sequence of 0/1-valued fair coin flips. In actual fact, the number of carry bits induced by one 1-bit in b could be one more, namely when the leftmost

end of the sequence of carry-bits coincides with another position where b has a 1-bit. The number of positions where this can happen is the random variable $|b \wedge c|$. Therefore the random variable $|c|$ is majorized² by the random variable $|b \wedge c| + S$, where $S = \sum_{i=1}^w G_i$ is the sum of w i.i.d. geometrically distributed random variables (each with parameter $1/2$, and support $\{0, 1, 2, \dots\}$).

Therefore we have

$$\begin{aligned} \mathbb{P}[\Delta(a, a + b) > 2w + s\sqrt{w}] &= \mathbb{P}[w + |c| - 2|b \wedge c| > 2w + s\sqrt{w}] \\ &= \mathbb{P}[|c| > 2|b \wedge c| + w + s\sqrt{w}] \leq \mathbb{P}[|c| > |b \wedge c| + w + s\sqrt{w}] \leq \mathbb{P}[S > w + s\sqrt{w}] \\ &= \mathbb{P}[\text{Bin}(2w + s\sqrt{w}, 1/2) < w] \leq \exp\left(-\frac{(s\sqrt{w})^2}{8w + 4s\sqrt{w}}\right) = \exp\left(-\frac{s^2}{8 + 4s/\sqrt{w}}\right). \end{aligned}$$

The second inequality uses majorization. The penultimate equality holds because the event ‘ $S > w + s\sqrt{w}$ ’ is the same as the event that in a sequence of $2w + s\sqrt{w}$ fair coin flips, there are fewer than w successes. We upper bound the probability of the latter by Chernoff’s inequality. \square

Heuristic 1. *The above lemma still holds when setting $a = hg_1$ and $b = f$ for f, g, h distributed as in the AJPS cryptosystem.*

Remark 1. The above heuristic is corroborated by experiments, see Appendix A. More concretely, for primes $n \leq 2000$, it holds that $\Delta(-g_2h, g_1h) \leq 2w - 1$ for more than half of the keys, and that $\Delta(-g_2h, g_1h) \leq 2w + 7$ for about 90% of the keys.

We are now ready to analyze the space and time complexity of the algorithm. We will set $\alpha = 1/2$. We can see immediately that the algorithm requires $\tilde{O}(|G_1^{(\alpha)}|) = \tilde{O}\left(\binom{n/2}{w/2}\right)$ space.

To analyze the time complexity, we first note that Step 3 of the algorithm costs $|G_1^{(\alpha)}|$ insertions into the data structure, so the cost is $\tilde{O}(|G_1^{(\alpha)}|) = \tilde{O}\left(\binom{n/2}{w/2}\right)$. The loop in Step 4 runs $|G_2^{(\alpha)}|$ times, and the iteration corresponding to some $g_2 \in G_2^{(\alpha)}$ costs approximately $1 + \ell(g_2)$, where $\ell(g_2) = |\{g_1 \in G_1^{(\alpha)} : \mathcal{H}(g_1h) = \mathcal{H}(-g_2h)\}|$. The total cost of Step 4 is thus at most:

$$|G_2^{(\alpha)}| + \sum_{g_2 \in G_2^{(\alpha)}} \ell(g_2).$$

We can rewrite the above as

$$\begin{aligned} |G_2^{(\alpha)}| + \sum_{v \in \mathbb{F}_2^B} |\{(g_1, g_2) \in G_1^{(\alpha)} \times G_2^{(\alpha)} : \mathcal{H}(g_1h) = \mathcal{H}(-g_2h) = v\}| \\ = |G_2^{(\alpha)}| + |\{(g_1, g_2) \in G_1^{(\alpha)} \times G_2^{(\alpha)} : \mathcal{H}(g_1h) = \mathcal{H}(-g_2h)\}|. \end{aligned}$$

² Random variable X majorizes random variable Y , if $\mathbb{P}[X > t] \geq \mathbb{P}[Y > t]$ for all t .

Heuristic 2. For every fixed $\mathcal{H} \in \mathcal{F}_B$, with high probability over g and f as chosen in the AJS system, we have $|\{(g_1, g_2) \in G_1^{(\alpha)} \times G_2^{(\alpha)} : \mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)\}| \approx |G_1^{(\alpha)}| \cdot |G_2^{(\alpha)}| 2^{-B}$.

Remark 2. The above heuristic is obtained by considering all $\mathcal{H}(g_1 h)$ and $\mathcal{H}(g_2 h)$ values as independent random uniform strings of B bits. The validity of this heuristic is confirmed by the experiments presented in Appendix A.1.

Let $g = g_1^* + g_2^*$ be a solution. The algorithm will only find this g if $\mathcal{H}(g_1^* h) = \mathcal{H}(-g_2^* h)$, in which case we say \mathcal{H} is *good* for g . By Lemma 2 and assuming Heuristic 1, $\Delta(g_1^* h, -g_2^* h) \leq 2w + s\sqrt{w}$ happens with high probability, for a fixed constant s . So, assuming $\Delta(g_1^* h, -g_2^* h) \leq 2w + s\sqrt{w}$, the hash function \mathcal{H} is good with probability at least $p(B) = \frac{\binom{n-2w-s\sqrt{w}}{B}}{\binom{n}{B}}$ where the probability is over the function family $\mathcal{F}_B = \{\mathcal{H}_B : |\mathcal{B}| = B\}$.

Lemma 3. Under the above heuristics, setting $\alpha = 1/2$ and $B = \lceil \log_2 \binom{n/2}{w/2} \rceil$ ($\approx \frac{w}{2} \log(n/w) + O(w)$), the time complexity of the algorithm is $\tilde{O}\left(\sqrt{\binom{n}{w}}\right)$.

Proof. Ignoring polylogarithmic factors, the complexity of the algorithm equals $|G_1^{(\alpha)}| + |G_2^{(\alpha)}| + |G_1^{(\alpha)}| \cdot |G_2^{(\alpha)}| 2^{-B}$. Note that $|G_2^{(\alpha)}| 2^{-B} \leq 1$, by the choice of B and the fact that $|G_2^{(\alpha)}| = \binom{n/2}{w/2}$. Therefore the complexity of steps 1–4 equals $2|G_1^{(\alpha)}| + |G_2^{(\alpha)}| = 3\binom{n/2}{w/2} = \tilde{O}\left(\sqrt{\binom{n}{w}}\right)$. To achieve constant success probability, we repeat the algorithm $1/p(B)$ times, which is, as we will show, polynomial in n . We use the identity $\ln \binom{m}{\ell} = \ell \ln(m/\ell) + \ell + O(\ln m)$ whenever $\ell = \tilde{O}(\sqrt{m})$, and the fact that $w^2 \approx n/4$. We have:

$$\begin{aligned} \ln \frac{1}{p(B)} &= \ln \binom{n}{B} - \ln \binom{n-2w-s\sqrt{w}}{B} = -B \ln \left(\frac{n-2w-s\sqrt{w}}{n} \right) + O(\ln n) \\ &= (1 + o(1)) \frac{2wB}{n} + O(\ln n) = (1 + o(1)) \frac{w^2}{n} \ln(n/w) + O(\ln n) = O(\ln n). \end{aligned}$$

□

4 Quantum Meet-in-the-Middle Attack

We now present our quantum meet-in-the-middle attack. The first example of a quantum meet-in-the-middle algorithm was the collision finding algorithm of Brassard et al. [9]. Similar ideas were later used in quantum algorithm for the subset sum problem [5], and a quantum attack on the NTRU cryptosystem [23], which have a similar structure to the algorithm presented here. One difference in our new algorithm is the use of Ambainis’s variable-cost quantum search algorithm [3], described in Sect. 2.2.

The algorithm presented in this section requires time and space $\tilde{O}\left(\binom{n}{w}^{1/3}\right)$. The bulk of the memory required for this quantum algorithm must be *quantum*

accessible, meaning it does not need to be able to store a quantum state, but must be accessible in superposition. Only $O(n)$ of the space used by the algorithm must be fully quantum memory, capable of being in an arbitrary superposition.

The quantum algorithm presented and analyzed in this section is then very similar to the classical MITM attack, except we use quantum search to search over all $g_2 \in G_2^{(\alpha)}$, and then since the complexity of this step of the algorithm decreases in the quantum case, it is optimal to use $\alpha = 1/3$ rather than $\alpha = 1/2$.

Detailed description of algorithm. Our quantum MITM attack proceeds as follows:

1. Choose a uniformly random $\mathcal{H} \in \mathcal{F}_B$.
2. Initialize an empty hash table \mathcal{D} .
3. For each $g_1 \in G_1^{(\alpha)}$:
 - (a) Insert $(g_1, \mathcal{H}(g_1h))$ into \mathcal{D} .
4. Quantum search (using Theorem 1) for $g_2 \in G_2^{(\alpha)}$ that satisfies the following checking procedure:
 - (a) Look up $\mathcal{H}(-g_2h)$ in \mathcal{D} , and let $L(g_2)$ be the resulting list of values g_1 such that $\mathcal{H}(g_1h) = \mathcal{H}(-g_2h)$.
 - (b) Quantum search for g_1 in L that satisfies the following checking procedure:
 - i. If $|g_1 + g_2| = w$ and $|(g_1 + g_2)h| = w$, then output 1.

Analysis of algorithm. We will use $\alpha = 1/3$. The algorithm requires $|G_1^{(\alpha)}| = \binom{\alpha n}{\alpha w} = \binom{n/3}{w/3}$ quantum accessible memory, and $O(\log |G_2^{(\alpha)}|) = O(n)$ quantum memory.

In order to upper bound the time complexity, we will make use of Heuristic 2 with $\alpha = 1/3$. Then we have the following.

Lemma 4. *Assuming Heuristics 1 and 2 with $\alpha = 1/3$, setting $B = \lceil \log_2 \binom{n/3}{w/3} \rceil$, the time complexity of the algorithm is $\tilde{O} \left(\binom{n}{w}^{1/3} \right)$.*

Proof. As in the classical algorithm, the time complexity of Steps 1 to 3 of the quantum algorithm is $\tilde{O}(|G_1^{(\alpha)}|) = \tilde{O} \left(\binom{\alpha n}{\alpha w} \right)$, which, in this case, is $\tilde{O} \left(\binom{n/3}{w/3} \right)$. For a particular $g_2 \in G_2^{(\alpha)}$, Steps 4a and 4b together cost (neglecting negligible factors) $1 + \sqrt{\ell(g_2)}$, so using variable cost quantum search, as described in Sect. 2.2, the total cost of Step 4 is

$$\sqrt{\sum_{g_2 \in G_2^{(\alpha)}} (1 + \sqrt{\ell(g_2)})^2} = O \left(\sqrt{|G_2^{(\alpha)}|} + \sqrt{\sum_{g_2 \in G_2^{(\alpha)}} \ell(g_2)} \right).$$

By 2 and the choice of B , we have $\sum_{g_2 \in G_2^{(\alpha)}} \ell(g_2) \approx |G_1^{(\alpha)}| \cdot |G_2^{(\alpha)}| 2^{-B} \leq |G_2^{(\alpha)}|$.

Thus, the total complexity of steps 1–4 of the attack is $O \left(\sqrt{|G_2^{(\alpha)}|} \right) =$

$O\left(\sqrt{\binom{2n/3}{2w/3}}\right) = \tilde{O}\left(\binom{n/3}{w/3}\right)$. Finally, as in Sect. 3, \mathcal{H} is *good* with probability $p(B) = \frac{\binom{n-2w-s\sqrt{w}}{B}}{\binom{n}{B}}$. To achieve constant success probability, we repeat $1/p(B)$ times, which is polynomial in n by a similar reasoning as in Lemma 3. \square

5 Analysis of the Beunardeau et al. Attack

Within a week of the publication of the AJPS cryptosystem [1], an experimental attack was proposed by Beunardeau et al. [6]. This attack exploits the fact that a certain lattice, derived from the public key of the AJPS cryptosystem and two well-chosen partitions, has very short vectors. One of these short vectors, which can be found by means of the LLL lattice reduction algorithm [20], represents the private key.

Although Beunardeau et al. do not give a clear asymptotic estimate of the complexity of their attack, they do suggest tentatively that it might run in time $2^{2w}n^{O(1)}$, where $w = \lfloor \sqrt{n}/2 \rfloor$ is the Hamming weight of secret key $g \in R$ [6, Sect. 2.2]. More specifically, once a partition is chosen, the attack runs in polynomial time $n^{O(1)}$, and the probability that it is successful should be about 2^{-2w} .

Remark 3. Note that this probability is taken only over the randomness of the secret key. It is not obvious that one can amplify the success probability for a fixed key up to a constant by repeating the attack with 2^{2w} different partitions. Indeed, there could be certain keys that are caught by a fraction of partitions significantly smaller than 2^{-2w} .

In this section, we propose an analysis of a simplified version of their attack. Using standard lattice heuristics we can argue that, for each pair of partitions, the probability that a secret key will be found by applying LLL on the derived lattice equals $(\frac{1}{2} - c(\frac{d}{w})^2 + o(1))^{2w}$, where d is the lattice dimension, and c is a very small constant, say $1/140$. The lattice dimension d corresponds to the number of blocks in a partition of the bits of f and g . While in theory we can choose d between 2 and $O(w)$, in order to find f and g for a particular h , we will generally need to choose d as large as $\Omega(w)$. We discuss this more at the end of Sect. 5.3. While asymptotically slightly different from the tentative conclusion of [6], this analysis certainly does not contradict the fact that this attack is quite efficient in practice, and remains the best known attack (better than our MITM attack).

We remark that one could also replace LLL with a perfect SVP-oracle to raise the success probability to $(\frac{1}{2} + o(1))^{2w}$, but this would increase the running time of the lattice reduction step to $2^{\Theta(d)}$. Namely, for partitions of size $d = \Theta(w)$ the ratio of the cost over success probability remains at least $2^{(2+\delta)w+o(w)}$ for a fixed $\delta > 0$.

Finally, we note that this attack can also be sped up with a quantum computer. If, for a particular fixed key g , the probability that a sampled partition allows the LLL subroutine to find the secret key is p , then there is a quantum

algorithm that finds the key in only $\sqrt{1/p}$ calls to the subroutine, compared to the $1/p$ calls required by a classical algorithm. So under the heuristic assumption that $p \approx 2^{-2w}$, there is a quantum algorithm that recovers the key in time $\approx 2^w$.

Unfortunately, despite some effort, we have not been able to answer the question left open by Beunardeau et al.: “Are there classes of public keys that are harder to recover using this lattice attack, and if so, which ones?”

5.1 Partitions

In this section, we show how partitioning of $[n] = \{0, \dots, n - 1\}$ can lead to a short representation of the secret key $g \in R = \mathbb{Z}/N\mathbb{Z}$. The overall idea is to write g as a binary string in \mathbb{F}_2^n , as before. Since g has a low Hamming weight, one can imagine the one-valued bits scattered sparsely among the n possible positions. One then chooses interval-like subsets of $[n]$ such that, with any luck, each one-valued bit falls in the right-half of one of these subsets. In that case, each subset of $[n]$ in the partition corresponds to a binary substring of g representing a “small” number. Consequently, the array of these numbers can be considered as a short representation of g . An example is depicted in Table 2.

Remark 4. Because of the bit-wise arithmetic in R , it is natural to consider interval-like partitions only. An interval-like partition P of $[n]$ consists of subsets that are of the form $\{a, a + 1, a + 2, \dots, b - 1, b\}$ for $a, b \in [n]$, i.e., subsets without ‘gaps’. Due to the fact that multiplication by 2^i in R simply shifts all binary representations by i , we also allow subsets of the form $\{a, a + 1, \dots, n - 2, n - 1, 0, 1, \dots, b\}$.

Remark 5. Formally, our approach is slightly different from the one of Beunardeau et al. [6]. Namely, they define partitions with black and white blocks, hoping that all 1-valued bits of the secret key fall into the white blocks of the partitions. It turns out, however, that the black blocks do not play any role in the construction of the lattice related to this partition. Therefore, we prefer to omit the black partitions in our approach. This alteration has no algorithmic impact and is merely an editorial choice simplifying the analysis.

Table 2. Example partition of g with Hamming weight 3.

g	0010000000001000010
Partition	$\{19, 18, 17, 16\}, \{15, 14, 13, 12\}, \{11, \dots, 6\}, \{5, \dots, 0\}$
g partitioned in a “good” way	0010 0000 000001 000010
Array of decimal numbers representing g	$[2, 0, 1, 2], g = 2 \cdot 2^{16} + 0 \cdot 2^{12} + 1 \cdot 2^6 + 2 \cdot 2^0$

5.2 Lattice Reduction

Lattice construction. Given any two interval-like partitions $P = \{P_1, \dots, P_k\}$, $Q = \{Q_1, \dots, Q_\ell\}$ of $[n]$ and a public key $h \in R$. Let p_i, q_i be the least elements of P_i, Q_i respectively. Then, one can consider the following lattice.

$$\mathcal{L}_{P,Q,h} = \left\{ (x_1, \dots, x_k, y_1, \dots, y_\ell) \mid h \cdot \sum_{i=1}^k 2^{p_i} \cdot x_i - \sum_{j=1}^{\ell} 2^{q_j} \cdot y_j \equiv 0 \pmod{N} \right\}.$$

This lattice $\mathcal{L}_{P,Q,h}$ has determinant $\Delta = N$ and dimension $d = k + \ell$. Namely, as $\mathcal{L}_{P,Q,h}$ is a sublattice of \mathbb{Z}^d , we have $\det(\mathcal{L}_{P,Q,h}) = \det(\mathbb{Z}^d) \cdot [\mathbb{Z}^d : \mathcal{L}_{P,Q,h}] = N$, since $\det(\mathbb{Z}^d) = 1$ and the group index equals N .

This lattice contains vectors of the form $(0, \dots, 0, 2^m, -1, 0, \dots, 0)$, for some m , which we will call ‘structural’ vectors. These structural vectors have length $\sqrt{4^{|P_i|} + 1}$ and $\sqrt{4^{|Q_i|} + 1}$. For example, $(2^{p_2 - p_1}, -1, 0, \dots, 0) \in \mathcal{L}_{P,Q,h}$ is a structural vector which is easily seen to have the described length, observing that $p_2 - p_1 = |P_1|$. Applying this example for every two subsequent variables of the same kind, one arrives at all structural vectors.

Definition 1 (Secret vector). Let $h = f/g \in R$ be as in the AJPS-cryptosystem, suppose $P = \{P_1, \dots, P_k\}$ and $Q = \{Q_1, \dots, Q_\ell\}$ are interval-like partitions of $[n]$ and denote $p_i = \min P_i$ and $q_j = \min Q_j$. We define the secret vector

$$s := (g_1, \dots, g_k, f_1, \dots, f_\ell) \in \mathcal{L}_{P,Q,h},$$

where $0 \leq g_i < 2^{|P_i|}$ and $0 \leq f_j < 2^{|Q_j|}$ are the unique natural numbers such that $\sum_{i=1}^k g_i \cdot 2^{p_i} = g$ and $\sum_{j=1}^{\ell} f_j \cdot 2^{q_j} = f$.

Remark 6. The vector s is actually just the concatenation of the vectors (g_1, \dots, g_k) and (f_1, \dots, f_ℓ) , which are constructed from g, P and f, Q respectively as in Table 2.

Applying LLL Let us recall the guarantees provided by the LLL algorithm.

Lemma 5 ([20, 21]). For any $\gamma > \sqrt{4/3}$, the LLL_γ -algorithm applied to a d -dimensional lattice L returns, within polynomial time, a basis (b_1, \dots, b_d) of L satisfying

- $\|b_1\| \leq HF(L) := \gamma^{(d-1)/2} \cdot \Delta_L^{1/d}$ (Hermite factor bound);
- $\|b_1\| \leq AF(L) := \gamma^{d-1} \cdot \lambda_1(L)$ (Approximation factor bound).

where $\lambda_1(L)$ is the length of a shortest nonzero vector of L , and Δ_L is the determinant of the lattice L .

In practice, LLL performs much better. For cryptanalytic purposes, one often assumes $\gamma = 1.04$, which is corroborated by many experiments [22]. In the current analysis, this practical value of γ will be used.

The inequalities in Lemma 5 give rise to two so-called *regimes* of LLL_γ , the *Hermite regime* and the *Approximation regime*. A lattice L lies in the Hermite regime when $\text{HF}(L) \leq \text{AF}(L)$, and lies in the Approximation regime whenever $\text{AF}(L) < \text{HF}(L)$. One distinguishes these two cases because the output of LLL differs significantly between the regimes. This effect is most prominent when a single, unique short vector causes a lattice to be in the Approximation regime; in that case LLL typically outputs this particular short vector [10, Sect. 3.3].

One would like to have that this last scenario holds for the lattice $\mathcal{L}_{P,Q,h}$ and the secret vector s . So, informally, one wishes to have no vectors in $\mathcal{L}_{P,Q,h}$ that are shorter than usual except for the secret vector s . One obstacle could be that the structural vectors are too short, causing s not to be unique. However, we will rule out this possibility by comparing the lengths of these structural vectors to the *Gaussian heuristic* of $\mathcal{L}_{P,Q,h}$.

The Gaussian heuristic uses a geometric argument to estimate the length of the shortest vector of a lattice [10]. For d -dimensional lattices L one expects $\lambda_1(L) \approx \sqrt{d/(2\pi e)} \cdot \Delta_L^{1/d}$, according to this heuristic. Applying this to the lattice of interest, one obtains $\lambda_1(\mathcal{L}_{P,Q,h}) \approx \sqrt{n/(2\pi e)} \cdot 2^{\frac{n}{d}}$. Recall that the structural vectors have approximate length $2^{|P_i|}$ and $2^{|Q_i|}$. So, whenever $|P_i|, |Q_i| > n/d + \Theta(\log n)$, we have $2^{|P_i|}, 2^{|Q_i|} > \sqrt{d/(2\pi e)} \cdot 2^{\frac{n}{d}}$. So, in this case, the structural vectors are not shorter than the estimate of the Gaussian heuristic and hence longer than the secret vector s . Note that the average size of $|P_i|, |Q_i|$ is $2n/d$, meaning that this constraint is not so restrictive.

Therefore, we assume the following heuristic.

Heuristic 3. *The attack of Beunarneau et al. is successful in recovering the secret vector s if s (as in Definition 1) is the shortest vector and causes the lattice $\mathcal{L}_{P,Q,h}$ to fall into the Approximation regime, i.e., $\text{AF}(\mathcal{L}_{P,Q,h}) < \text{HF}(\mathcal{L}_{P,Q,h})$.*

From the above heuristic we can deduce that the lattice attack succeeds if

$$\|s\| \cdot \gamma^{d-1} < \gamma^{(d-1)/2} \cdot 2^{n/d} = \text{HF}(\mathcal{L}_{P,Q,h}).$$

Moreover, according to the study of Albrecht et al. [2] on the behavior of LLL for unique-SVP instances, this condition should be essentially tight. More precisely, we expect the attack to fail with overwhelming probability when $\text{AF}(\mathcal{L}_{P,Q,h}) > O(\sqrt{d}) \cdot \text{HF}(\mathcal{L}_{P,Q,h})$.

The metric bounds $\|s\|_\infty \leq \|s\| \leq \sqrt{d} \cdot \|s\|_\infty$ imply that the attack passes when $\sqrt{d} \cdot \|s\|_\infty < \gamma^{-(d-1)/2} \cdot 2^{\frac{n}{d}}$ and is expected to fail when $\|s\|_\infty > O(\sqrt{d}) \cdot \gamma^{-(d-1)/2} \cdot 2^{\frac{n}{d}}$. Since $\|s\|_\infty = \max\{g_i, f_i\}$, we can write $\|s\|_\infty = 2^r$, where r is the bit size of the maximum of the g_i and f_i . Putting this in the inequalities and taking base-two logarithms, yields the following. The attack succeeds whenever $r < \frac{n}{d}(1 - \delta_1 - \delta_2)$ and is expected to fail when $r > \frac{n}{d}(1 - \delta_1 + \delta_2 + O(d/n))$, where

$$\delta_1 = \frac{d(d-1) \cdot \log_2(\gamma)}{2n} \quad \text{and} \quad \delta_2 = \frac{d \cdot \log_2(d)}{2n}.$$

5.3 Success Probability Analysis

Let P and Q be partitions with block sizes at least $n/d + \Theta(\log n)$, where $d = k + \ell$ with $k = |P|$ and $\ell = |Q|$. We analyze the success probability of the lattice attack with respect to random f and $g \in R$ both of Hamming weight $w = \lfloor \sqrt{n}/2 \rfloor$.

From the previous section, we found that it suffices that the non-zero bits of f and g fall in the rightmost r bits of each block, in order to make LLL find the secret vector. So, for g , the total number of bits that are allowed to be one equals $k \cdot r$. Therefore, we can approximate the probability of the bits of f and g all falling in the good region by

$$\left(\frac{\ell \cdot r}{n}\right)^w \left(\frac{k \cdot r}{n}\right)^w.$$

Putting in the upper and lower bound for r , we obtain an upper and lower bound for the success probability p of the attack.

$$\left(\frac{\ell k(1 - \delta_1 - \delta_2)^2}{d^2}\right)^w < p < \left(\frac{\ell k(1 - \delta_1 + \delta_2 + O(d/n))^2}{d^2}\right)^w.$$

In order to maximize the above probability, we will assume that $k = \ell = d/2$ and $d = O(w)$. Namely, the fraction $\frac{\ell k}{(\ell+k)^2}$ attains its maximum at $\ell = k = d/2$.

Recalling $w^2 \approx n/4$, we obtain $\delta_1 = \frac{d(d-1)}{2n} \cdot \log_2(\gamma) \approx \frac{1}{8} \left(\frac{d}{w}\right)^2 \cdot \log_2(\gamma)$ and $\delta_2 = o(1)$ as $n \rightarrow \infty$. Therefore

$$\left(\frac{1 - \delta_1 - o(1)}{2}\right)^{2w} < p < \left(\frac{1 - \delta_1 + o(1)}{2}\right)^{2w}.$$

Thus, assuming Heuristic 3, the success probability of LLL recovering a randomly chosen AJPS secret key pair $(f, g) \in R^2$ from the lattice $\mathcal{L}_{P,Q,h}$ (where $h = f/g$), is roughly $\left(\frac{1}{2} - c \left(\frac{d}{w}\right)^2 + o(1)\right)^{2w}$, where $c = \log_2(\gamma)/8 = \log_2(1.04)/8 \approx 1/140$. This probability value suggests that one should start with partitions with a small number of blocks, exploiting both the low dimension m of the lattice $\mathcal{L}_{P,Q,h}$ and a slightly larger success probability. Note, however, that it is not likely that the secret key $s = (g, f)$ will be recovered in this stage; the smaller d is, the fewer possible partitions there are, so the need to sample new partitions will require us to increase d to $\Omega(w)$ for most keys.

Replacing LLL by an SVP-oracle. If one replaces LLL by an SVP-oracle, the success condition from Heuristic 3 needs to be amended. Instead, the attack would be successful when s is the shortest vector of L . Heuristically this is the case if and only if s is shorter than what is predicted by the Gaussian Heuristic $\lambda_1(L) \approx \sqrt{d/2\pi e} \cdot \Delta_L^{1/d}$. Using a similar analysis, this leads to a success probability of $2^{-2w+o(1)}$. Note however that the best SVP-solvers [4] need time $(3/2)^{d/2}$, which would increase the overall complexity of the attack significantly.

5.4 Generalization to Scaled Partitions

The attack that is treated above is a simplification of the attack of Beunardeau et al.; essentially we omitted a ‘scaling’ technique [6, Sect. 2.2, ‘Trying partitions’]. This particular technique allows the variation of partition sizes and the fraction of each partition block that must consist of leading 0s.

The lattice $\mathcal{L}_{P,Q,h}$ scaled by the vector $\sigma = (\sigma_1, \dots, \sigma_k, \sigma'_1, \dots, \sigma'_\ell) \in \mathbb{R}^d$ can be defined explicitly as follows.

$$\mathcal{L}_{P,Q,h}^\sigma = \left\{ (\sigma_1 x_1, \dots, \sigma_k x_k, \sigma'_1 y_1, \dots, \sigma'_\ell y_\ell) \mid h \cdot \sum_{i=1}^k 2^{p_i} \cdot x_i - \sum_{j=1}^{\ell} 2^{q_j} \cdot y_j \equiv 0 \pmod{N} \right\}.$$

Allocating less weight σ_i to the content x_i of a certain partition P_i lets a lattice reduction algorithm tolerate larger values x_i ; this means that the required fraction of leading 0s in this partition is diminished. This technique implies more freedom in choosing block sizes and required fractions of leading 0s.

Note, however, that scaling the entire lattice $L \mapsto cL$ by a constant won’t affect the attack at all. Therefore, one might require, without loss of generality, that $\prod_{i=1}^k \sigma_i \prod_{j=1}^{\ell} \sigma'_j = 1$. This implies that the increase and decrease of the fractions of leading 0s of the blocks are in an equilibrium, not affecting the total region where non-zero bits are allowed.

So, this extension possibly increases the number of public keys that can be broken but does not affect the running time nor the success probability of the attack. Even considering this generalization, we were not able to prove that this improved attack could recover *every* key with constant probability in time $2^{(2+\delta)w+o(1)}$ for some small constant $\delta > 0$.

Acknowledgments. The authors wish to thank David Naccache, Antoine Joux and Marc Beunardeau for helpful discussions, and the anonymous PQCrypto reviewers for useful feedback. LD is supported by a NWO Veni Innovational Research Grant under project number 639.021.645. SJ is supported by an NWO WISE Grant and an NWO Veni Innovational Research Grant under project number 639.021.752. RdW is partially supported by ERC Consolidator Grant 61530-QPROGRESS.

A Experiments

Since our MITM attack is not fully provable due to the presence of Heuristics 1 and 2, we provide some experimental verifications. The python scripts of those experiments are available at <https://github.com/lducas/MiTM-Mersenne>.

One tweak in our implementation is that when w is odd, we do not split our space exactly into two equal parts. Instead we choose $w_1 = \lfloor w/2 \rfloor$, $w_2 = w - w_1$, and then choose n_1, n_2 , such that $\binom{n_1}{w_1} \approx \binom{n_2}{w_2}$. We will also simulate the quantum case, and choose $w_1 = \lceil w/3 \rceil$, $w_2 = w - w_1$, and then choose n_1, n_2 , such that $\binom{n_1}{w_1}^2 \approx \binom{n_2}{w_2}$. In both the classical and quantum case, we set $B = \lfloor \log_2 \binom{n_1}{w_1} \rfloor$.

A.1 Verification of Heuristic 2

We recall that Heuristic 2 states that the number of collisions $c = |\{(g_1, g_2) \in S_1 \times S_2 : \mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)\}|$ is approximately given by $c' = |S_1| \cdot |S_2| 2^{-B}$. We measure the ratio $r = c/c'$ experimentally, over 100 samples for each dimension n . Infrequently, this ratio may get as large as 3, yet for 90% of the experiments, it was very close to 1. Figure 1 below shows the 9th decile of r as n grows.

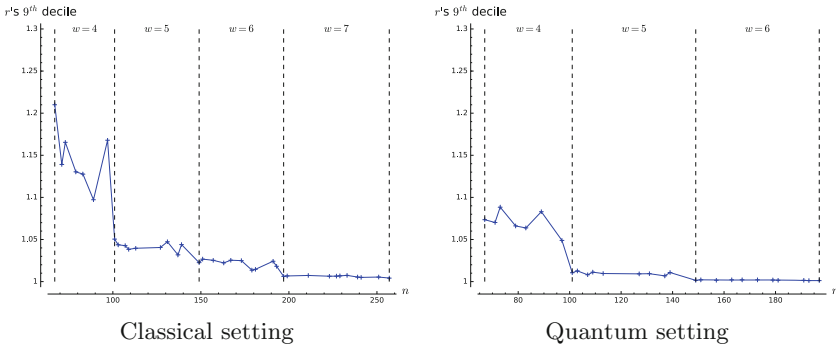


Fig. 1. 9th decile of the ratio between the measured number of collisions c and expected number of collisions c' according to Heuristic 2, over 100 experiments per dimension.

A.2 Running time and success probability

In Figs. 2 and 3, we report on the practical efficiency of our attack and compare it to our heuristic prediction. Note that in the quantum regime, the success probability of this MITM attack in practice is sometimes significantly larger than the theoretical prediction. This is most likely due to the fact that our

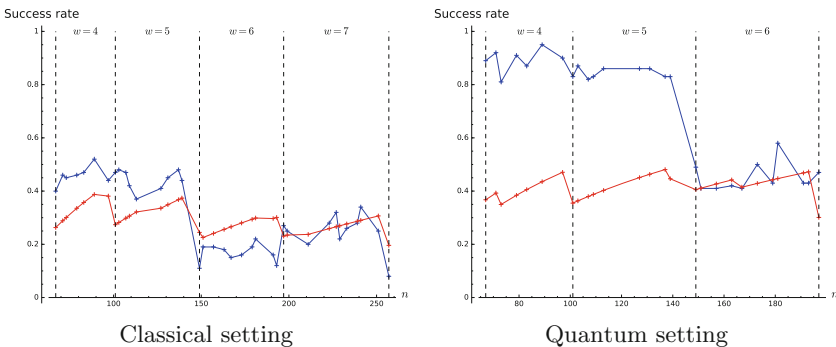


Fig. 2. Success rate of the attack over 100 trials (in blue), compared to the theoretical success rate $(1 - 2w/(n - B))^B$ (in red). The rather discontinuous shape of the red curve is due to the rounding of $w = \lfloor \sqrt{n}/2 \rfloor$ and $B = \lfloor \log_2 \binom{n_1}{w_1} \rfloor$. (Color figure online)

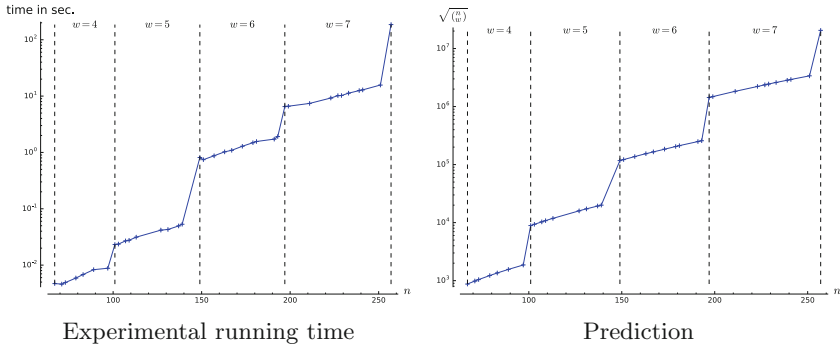


Fig. 3. Average running time of the classical attack over 100 trials in comparison with the function $\sqrt{\binom{n}{w}}$, which is the dominant factor in our asymptotic complexity.

analysis is done for one particular solution, while certain rotations of the same key may be found as well if its bits are properly balanced with respect to the split $\mathbb{F}_2^n = G_1 \oplus G_2$.


References

1. Aggarwal, D., Joux, A., Prakash, A., Santha, M.: A new public-key cryptosystem via Mersenne numbers. Cryptology ePrint Archive, Report 2017/481 (2017). <http://eprint.iacr.org/2017/481>
2. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving uSVP and applications to LWE. Cryptology ePrint Archive, Report 2017/815 (2017). <https://eprint.iacr.org/2017/815>
3. Ambainis, A.: Quantum search with variable times. Theory Comput. Syst. **47**(3), 786–807 (2010)
4. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Proceedings of 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016), pp. 10–24 (2016)
5. Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum algorithms for the subset sum problem. In: Proceedings of 5th International Conference on Post-Quantum Cryptography (PQCrypto 2013), pp. 16–33 (2013)
6. Beunardeau, M., Connolly, A., Géraud, R., Naccache, D.: On the hardness of the Mersenne low Hamming ratio assumption. In: Progress in Cryptology - LATIN-CRYPT 2017 (2017). <http://eprint.iacr.org/2017/522>
7. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschr. Phys. **46**(4–5), 493–505 (1998)
8. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum Computation and Quantum Information: A Millennium. AMS Contemporary Mathematics Series Millennium, vol. 305, pp. 53–74. AMS (2002)
9. Brassard, G., Høyer, P., Tapp, A.: Quantum algorithm for the collision problem. ACM SIGACT News **28**, 14–19 (1997). [arXiv:quant-ph/9705002](https://arxiv.org/abs/quant-ph/9705002)

10. 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
11. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of 28th Annual ACM Symposium on the Theory of Computing (STOC 1996), pp. 212–219 (1996)
12. Hellman, M.: A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory* **26**(4), 401–406 (1980)
13. 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>
14. Howgrave-Graham, N.: A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 150–169. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_9
15. Howgrave-Graham, N., Silverman, J.H., Whyte, W.: A meet-in-the-middle attack on an NTRU private key. Technical report, NTRU Cryptosystems, June 2003
16. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of 30th Symposium on Theory of Computing (STOC 1998) (1998)
17. Laarhoven, T.: Search problems in cryptography. Ph.D. thesis, Eindhoven University of Technology (2015). <http://www.thijs.com/docs/phd-final.pdf>
18. Laarhoven, T.: Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 3–22. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_1
19. Laarhoven, T., Mosca, M., van de Pol, J.: Finding shortest lattice vectors faster using quantum search. *Des. Codes Crypt.* **77**(2–3), 375–400 (2015)
20. Lenstra, A.K., Lenstra, H.W., Lovasz, L.: Factoring polynomials with rational coefficients. *Math. Ann.* **261**, 515–534 (1982)
21. Nguyen, P.Q.: Hermite’s constant and lattice algorithms. In: Nguyen, P., Vallée, B. (eds.) The LLL Algorithm. ISC, pp. 19–69. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02295-1_2
22. Nguyen, P.Q., Stehlé, D.: LLL on the average. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 238–256. Springer, Heidelberg (2006). https://doi.org/10.1007/11792086_18
23. Wang, H., Ma, Z., Ma, C.: An efficient quantum meet-in-the-middle attack against NTRU-2005. *Chin. Sci. Bull.* **58**, 3514–3518 (2013)



Implementing Joux-Vitse's Crossbred Algorithm for Solving \mathcal{MQ} Systems over \mathbb{F}_2 on GPUs

Ruben Niederhagen^{1(✉)}, Kai-Chun Ning^{2(✉)}, and Bo-Yin Yang^{3(✉)} 

¹ Fraunhofer SIT, Darmstadt, Germany
ruben@polycephaly.org

² Eindhoven University of Technology, Eindhoven, The Netherlands
kaichun.ning@gmail.com

³ IIS and CITI, Academia Sinica, Taipei, Taiwan
by@crypto.tw

Abstract. The hardness of solving multivariate quadratic (\mathcal{MQ}) systems is the underlying problem for multivariate-based schemes in the field of post-quantum cryptography. The concrete, practical hardness of this problem needs to be measured by state-of-the-art algorithms and high-performance implementations. We describe, implement, and evaluate an adaption of the Crossbred algorithm by Joux and Vitse from 2017 for solving \mathcal{MQ} systems over \mathbb{F}_2 . Our adapted algorithm is highly parallelizable and is suitable for solving \mathcal{MQ} systems on GPU architectures. Our implementation is able to solve an \mathcal{MQ} system of 134 equations in 67 variables in 98.39 hours using one single commercial Nvidia GTX 980 graphics card, while the original Joux-Vitse algorithm requires 6200 CPU-hours for the same problem size. We used our implementation to solve all the Fukuoka Type-I \mathcal{MQ} challenges for $n \in \{55, \dots, 74\}$. Based on our implementation, we estimate that the expected computation time for solving an \mathcal{MQ} system of 80 equations in 84 variables is about one year using a cluster of 3600 GTX 980 graphics cards. These parameters have been proposed for 80-bit security by, e.g., Sakumoto, Shirai, and Hiwatari at Crypto 2011.

Keywords: Post-quantum cryptography
Multivariate quadratic systems · Parallel implementation · GPU

1 Introduction

With the advent of quantum computing, an adversary can efficiently break universally adopted public-key cryptographic schemes, e.g. RSA and elliptic-curve cryptography, with a sufficiently large quantum computer [16, 17]. In order to

This work is based on Kai-Chun Ning's master thesis under the supervision of Ruben Niederhagen, Tanja Lange, and Daniel J. Bernstein. Date: 2018.01.23. Permanent ID of this document: [f9066f7294db4f2b5fbd3e791fed78e](https://doi.org/10.1007/978-3-319-79063-3_6).

mitigate this imminent threat, cryptographic schemes that are resistant against quantum computers have drawn great attention from academia. These schemes are collectively referred to as post-quantum cryptography (PQC).

One potential candidate for PQC is *multivariate cryptography*. Multivariate cryptography relies on the difficulty of solving a system of m polynomial equations in n variables over a finite field. The complexity of solving a multivariate polynomial system (\mathcal{MP} problem) or a multivariate quadratic system (\mathcal{MQ} problem) where coefficients of the monomials are independently and uniformly distributed (i.e. random) is well-known to be NP-hard. An arbitrary \mathcal{MP} system can be transformed into an equivalent \mathcal{MQ} system by substituting monomials of degree larger than two with new variables and introducing extra equations to the system. Furthermore, a polynomial system over any extension field \mathbb{F}_{2^n} can be reduced into an equivalent system over \mathbb{F}_2 using Weil descent.

Since the early 1980s, various asymmetric multivariate encryption schemes (e.g., [5, 14, 18]) based on Hidden Field Equations (HFE) [10] as well as signature schemes (e.g., [6, 12, 13]) have been proposed. Besides these asymmetric schemes, some symmetric encryption schemes, e.g., the stream cipher QUAD [1], have been proposed and analyzed [20].

Introducing a trapdoor into an \mathcal{MQ} system for the use in public-key cryptography results in a system that is not truly random and typically exhibits a hidden structure that often can be exploited in its cryptanalysis. However, we do not focus on the cryptanalysis of any particular cryptographic scheme by exploiting some hidden structure. Our goal is to investigate the concrete, practical hardness of the underlying problem of solving random \mathcal{MQ} systems over \mathbb{F}_2 by providing an efficient, parallel implementation of the state-of-the-art algorithm.

This paper is structured as follows: In Sect. 2, we introduce the Crossbred algorithm by Joux and Vitse and our adaption to this algorithm. In Sect. 3, we describe our implementation of the adapted algorithm for a cluster of GPUs. In Sect. 4 we describe how to choose the parameters for our implementation, given a specific \mathcal{MQ} system size, and in Sect. 5, we provide an evaluation of our implementation.

The source code of our implementation and further information are available at www.polycephaly.org/mqsolver/.

2 Joux-Vitse’s Crossbred Algorithm

There are several approaches for solving \mathcal{MP} systems, e.g., Faugère’s F4 and F5 algorithms [7, 8] based on the computation of Gröbner-bases and a family of algorithms based on extended linearization (XL) [19]. For \mathcal{MQ} systems over \mathbb{F}_2 , Fast Exhaustive Search (FES) [3], i.e., efficient enumeration over the search space, was the approach used by the previous record holder [4] of Fukuoka MQ Type-I and Type-IV challenges¹. The record on Type-I challenges is now held by an implementation of the Crossbred algorithm by Joux and Vitse [11].

¹ <https://www.mqchallenge.org/>.

The basic idea of the XL algorithm is to extend the original \mathcal{MQ} system by multiplying it by all monomials up to a certain degree $D - 2$ and by treating monomials in the resulting degree- D system as linear variables. Solving this linear system gives a solution for the original \mathcal{MQ} system with high probability, if D is chosen large enough.

FES works by enumerating all possible assignments of the variables and by checking the correctness of each assignment with the original \mathcal{MQ} system. In contrast to a plain brute-force search, the possible assignments are enumerated in Gray-code order such that there is only one single variable with a different assignment in each enumeration step. This allows to compute the new evaluation result efficiently based on the change in regard to the previous evaluation result, which requires storage and recursive update of partial derivatives up to the total degree of the system [4].

2.1 The Crossbred Algorithm

The basic idea of Joux and Vitse's Crossbred algorithm is to extend the original \mathcal{MQ} system to a system with a degree D lower than the degree required for XL and to derive a sub-system that has at most degree d in the first k variables. This sub-system is then solved by iterating over the remaining $n - k$ variables and solving the resulting degree- d system in k variables in each iteration. For $d = 1$, this requires to only solve a *linear* system in k variables for each assignment of $n - k$ variables.

For example, by fixing the last two variables x_3 and x_4 , the sub-system

$$\mathcal{S} = \begin{cases} x_1x_4 + x_2x_3 + x_1 + x_3 + x_4 = 0 \\ x_1x_3 + x_3x_4 + x_2 + 1 = 0 \\ x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_4 = 0 \end{cases}$$

becomes a linear system in x_1 and x_2 . Clearly, the resulting linear system can be directly solved with Gaussian elimination, with which solutions to the system \mathcal{S} can be derived efficiently.

For a monomial $x^\alpha = x_1^{\alpha_1}x_2^{\alpha_2} \dots x_k^{\alpha_k}x_{k+1}^{\alpha_{k+1}} \dots x_n^{\alpha_n}$, the total degree of the first k variables is denoted as $\deg_k x^\alpha = \sum_{i=1}^k \alpha_i$. Given an \mathcal{MQ} system \mathcal{F} , the Crossbred algorithm first computes a degree- D Macaulay matrix with respect to a monomial order $>_{\deg_k}$ where monomials are sorted according to \deg_k in descending order. Subsequently the algorithm extracts at least k equations where the monomials of \deg_k larger than one (which are non-linear in x_1, \dots, x_k) are eliminated and only keeps monomials of $\deg_k \leq 1$ (which are linear in x_1, \dots, x_k). These equations give a sub-system that can be transformed into a linear system in the first k variables by fixing the remaining $n - k$ variables. After one such sub-system \mathcal{S} is obtained, Crossbred performs exhaustive search by fixing the last $n - k$ variables and testing whether or not the resulting linear system \mathcal{S}' is solvable. If so, solutions to \mathcal{S}' are checked with the original \mathcal{MQ} system \mathcal{F} . The algorithm terminates if a solution is found, otherwise it fixes $n - k$ variables in \mathcal{S} with another set of values and continues the exhaustive search procedure.

Algorithm 1. The Original Crossbred Algorithm

```

1: procedure CROSSBRED
2:   Input:
3:     an  $\mathcal{MQ}$  system of  $m$  equations in  $n$  variables  $\mathcal{F} = \{f_1, f_2, \dots, f_m\}$ 
4:     Macaulay degree:  $D$ 
5:     number of variables to keep:  $k$ 
6:     number of variables to fix during  $\mathcal{MQ}$  external hybridization:  $p$ 
7:
8:   for each  $(x_{n-p+1}, \dots, x_n)$  in  $\{0, 1\}^p$  do
9:     1. Fix the last  $p$  variables in  $\mathcal{F}$  to obtain an  $\mathcal{MQ}$  system  $\mathcal{F}'$ .
10:    2. Compute the degree- $D$  Macaulay matrix  $\text{Mac}_D^k$ 
11:       where monomials are sorted by  $\deg_k$  based on  $\mathcal{F}'$ .
12:    3. Extract  $r$  linearly independent equations  $\mathcal{S} = \{s_1, s_2, \dots, s_r\}$  from  $\text{Mac}_D^k$ 
13:       where monomials of  $\deg_k > 1$  have been eliminated.
14:
15:    Call FastEvaluate( $\mathcal{S}, k, n - p$ ) and
16:    for each output linear system  $\mathcal{S}'$  do
17:      4. Test if  $\mathcal{S}'$  is solvable. If so, extract solutions and verify them with  $\mathcal{F}$ .
18:      5. Continue if no solution is found.
19:    Otherwise output the solution and terminate.
20:    end for
21:  end for
22: end procedure

```

To obtain a linear system \mathcal{S}' from the extracted sub-system \mathcal{S} , the Crossbred algorithm uses a recursive algorithm called **FastEvaluate** to fix $n - k$ variables in \mathcal{S} . The basic idea of this algorithm is to split each polynomial into two groups of monomials. An arbitrary polynomial p can be written as $p = p_0 + x_i p_1$, where $x_i p_1$ are monomials that involve a specific variable x_i while p_0 are monomials that do not. It is clear from this form that p_0 is exactly the result of fixing $x_i = 0$ in p and $p_0 + p_1$ is the result of fixing $x_i = 1$ in p . This idea can be applied recursively to fix $n - k$ variables.

One can further fix some variables in the original \mathcal{MQ} system before computing Macaulay matrices, which is referred to as *external hybridation* by the authors [11]; here, we use the term *external hybridization*. The authors of the Crossbred algorithm consider external hybridization merely as a method to distribute the workload between computers and do not expect it to be asymptotically useful [11]. Nevertheless, this technique can be helpful to increase the number of variables that can be kept for linearization, which reduces the runtime of the algorithm significantly.

2.2 Adapting the Crossbred Algorithm for Parallel Implementation

The **FastEvaluate** algorithm proposed by Joux and Vitse has the disadvantage that computing the subsets p_0 and p_1 on higher levels of the recursion is relatively

expensive. We propose to use Gray-code enumeration [3] instead of FastEvaluate, which requires only $\mathcal{O}(2^{n-k} \cdot D \cdot k)$ machine instructions on the cost of $\mathcal{O}(\sum_{i=0}^D \binom{n-k}{i} \cdot k)$ memory.

Gray-code enumeration was proposed to efficiently evaluate a polynomial function $f(x_1, x_2, \dots, x_n)$ in all points $(x_1, x_2, \dots, x_n) \in \mathbb{F}_2^n$. To obtain the result of evaluating f on the next point $\mathbf{a} \in \mathbb{F}_2^n$ from the current result $f(\mathbf{a}')$ where only the i^{th} coordinates of \mathbf{a} and \mathbf{a}' differ, $\mathcal{O}(1)$ machine instructions are executed to combine $f(\mathbf{a}')$ with the result of evaluating the first order partial derivative $\frac{\partial f}{\partial x_i}$ on \mathbf{a}' [3]. In particular, $f(\mathbf{a}) = f(\mathbf{a}') + \frac{\partial f}{\partial x_i}(\mathbf{a}')$. This technique can be applied recursively to evaluate $\frac{\partial f}{\partial x_i}(\mathbf{a}')$ and its higher order partial derivatives until the partial derivative reduces to a constant. Therefore, if f is of degree D , $\mathcal{O}(D)$ operations are required to compute $f(\mathbf{a})$.

The same technique can also be applied to evaluate a function f whose output is a *linear function* in k variables instead of a constant over \mathbb{F}_2 by simply splitting the polynomial into a sum of $k+1$ sub-polynomials, one for each of the k variables and one for a constant term. For example, the polynomial

$$\begin{aligned} f &= x_1x_4x_5x_6 + x_1x_4x_5x_7 + x_4x_5x_6x_7 + x_1x_4x_5 + x_3x_4x_7 + x_3x_5 \\ &+ x_2x_4x_6 + x_4x_6x_7 + x_1x_4 + x_1x_5 + x_5x_7 + x_6x_7 + x_1 + x_2 + x_4 + 1 \end{aligned}$$

which is linear in x_1, x_2 , and x_3 can be split into the 4 polynomials

$$\begin{aligned} f_1 &= x_1(x_4x_5x_6 + x_4x_5x_7 + x_4x_5 + x_4 + x_5 + 1), \\ f_2 &= x_2(x_4x_6 + 1), \\ f_3 &= x_3(x_4x_7 + x_5), \\ f_4 &= x_4x_5x_6x_7 + x_4x_6x_7 + x_5x_7 + x_6x_7 + x_4 + 1, \end{aligned}$$

such that $f = f_1 + f_2 + f_3 + f_4$. Now, f can be evaluated by applying Gray-code enumeration to f_1, f_2, f_3 , and f_4 individually.

Since the result of evaluating f or any of its partial derivatives on a point $\mathbf{a} \in \mathbb{F}_2^4$ is a linear function that can be represented by four \mathbb{F}_2 elements (three variables and the constant term) and the last order partial derivatives reduce to constants, evaluating $f(\mathbf{a})$ takes at most $3 \cdot (3+1) + 1$ xor-operations and another $4 \cdot 2$ operations for computing the indices of the coordinates that changed during enumeration. In general, for a polynomial function f of degree D whose output is a linear function in k variables, evaluating f requires $\mathcal{O}(D \cdot k)$ operations.

Since a machine instruction operates on machine words, which for example have size 64 for 64-bit architectures or 32 on GPUs, multiple polynomials can be evaluated with Gray-code enumeration simultaneously. Therefore, the algorithm described above can be applied to fix $n - k$ variables in an extracted sub-system \mathcal{S} of m equations in n variables using $\mathcal{O}(D \cdot k)$ instructions, as long as m is not larger than the machine word size.

Gray-code enumeration can be easily parallelized: To run the enumeration with 2^t threads in parallel, first fix t variables in the sub-system \mathcal{S} with all t -tuples in $\{0, 1\}^t$ to create 2^t smaller sub-systems in $n - t$ variables. With this

approach, although the sub-systems are distinct from each other, their last order partial derivatives with respect to the $n - t - k$ variables that must be fixed are identical.

3 Implementation

Our target platform for the implementation is a hybrid cluster of workstations equipped with GPUs. Therefore, we have two processor architectures to our disposal: AMD64 CPUs and Nvidia GPUs (Kepler and Maxwell microarchitecture). The Gray-code enumeration part of the Crossbred algorithm is particularly easy to parallelize and therefore suitable for GPU deployment. Thus, we use the CPUs to generate and process the Macaulay matrix and the GPUs for Gray-code enumeration and linear-system solving.

3.1 Macaulay-Matrix Computations

The first step in Joux-Vitse's Crossbred algorithm is to extend the original \mathcal{MQ} system to a Macaulay matrix of degree D . (Our implementation works for $D = 3$ and $D = 4$.) The columns are ordered such that the monomials with $\deg_k > 1$ are in the front. Then, several (in our implementation 32) non-trivial vectors in the left kernel of the Macaulay matrix are computed. Finally, a sub-system linear in x_1, \dots, x_k is extracted for Gray-code enumeration.

Since the Macaulay matrix is very sparse, a sparse-system solver like the block Lanczos algorithm or the block Wiedemann algorithm could be used. However, the Macaulay matrix exhibits a special structure: Since the Macaulay matrix is generated from the original system by multiplying the polynomials with all monomials up to a certain degree, the resulting matrix is close to being diagonal. Therefore, we decided to exploit this special structure in a specifically adapted implementation of Gaussian elimination.

The first step is to compute the reduced echelon form of the original input system. This is a very small computation and requires a negligible amount of time. Then, we compute the Macaulay matrix \mathcal{M} such that the columns are in the required order. We store \mathcal{M} in a sparse representation. Then we search for rows in the Macaulay matrix that have an increasing number of leading zeros and swap them into place: Find a row that has no leading zeros and swap it to the top, find a row that has one leading zero and swap it to the second row, and so on. Due to the structure of the Macaulay matrix, usually about two thirds of the rows of the upper-triangular form of \mathcal{M} can be obtained just by swapping in suitable rows. Now, only the remaining one third of the upper-triangular form of \mathcal{M} needs to be computed. Observe that up to this point, \mathcal{M} can be stored in a sparse format and no costly row reductions needed to be performed.

In order to compute the remaining rows of the upper-triangular form of \mathcal{M} , one must perform row reduction. Therefore, we switch over to a dense representation by first performing row reduction on rows that have not found their final position during row-swapping with those that did. In this manner, we drop those

rows and columns that already have been pivoted by row-swapping and obtain a dense, reduced matrix \mathcal{RM} . On this matrix, we perform classical Gaussian elimination in order to compute the desired sub-system that is linear in x_1, \dots, x_k .

After \mathcal{RM} is computed, it can be copied to the GPU if the off-chip memory is large enough to accommodate it. Subsequently a sub-system can be extracted with Gaussian elimination on the GPU and copied back to the system main memory. On the other hand, if the size of \mathcal{RM} is too large or if the overall workload is pipelined between CPU and GPU, Gaussian elimination is simply performed on the CPU. We parallelized the CPU implementation using the *POSIX Thread API* to distribute the workload over all CPU cores. We observed during experiments that our GPU implementation on a Nvidia GTX 980 graphics card outperforms our CPU version on a AMD FX-8350 4 GHz processor by a factor of 9 in most cases.

Since the size of registers on a GPU is 32 bits and both Gray-code enumeration and linear system solving require the input system to be stored in column-wise format, only 32 linearly independent equations need to be extracted from the reduced Macaulay matrix \mathcal{RM} for the sub-system \mathcal{S} .

3.2 Fixing Variables in the Sub-system

We implemented the Gray-code enumeration algorithm for fixing $n - k$ variables in the degree- D sub-system \mathcal{S} to enumerate linear systems in k variables for the GPU architecture. The data structures used by Gray-code enumeration are allocated from the off-chip global memory. We simply distribute the workload over 2^t threads by fixing t variables in \mathcal{S} to obtain individual and independent smaller sub-systems $\mathcal{S}_i, 1 \leq i \leq 2^t$ for each thread. Since the last partial derivatives are constants and remain the same for all 2^t smaller sub-systems as noted in Sect. 2.2, they can be shared by all threads. Since they are constant, we store them in read-only constant memory.

The GPU threads in a warp begin enumeration with the same starting point and consequently they will access partial derivatives in the same order in each iteration. Therefore, the data structures for one warp can be interleaved to obtain optimal memory throughput. In addition, because of the cyclic nature of Gray-code enumeration, the last-level derivatives stored in constant memory are likely to be cached in the constant memory cache. Since the data of the 32 equations in the sub-system is stored in column-wise format, in total $\binom{n-k-t}{D}$ 32-bit integers are required for storing the constant last-level derivatives.

As described in Sect. 2.2, the evaluation of a k -linear polynomial is split into the evaluation of $k + 1$ polynomials. Therefore, we store the data for the non-constant partial derivatives for the 32 threads in one warp in basic units of $32(k+1)$ words interleaved in memory. Since for each of the $k + 1$ polynomials $n - k - t$ variables have to be fixed during enumeration, storing results of evaluating the non-constant partial derivatives of \mathcal{S}_i requires $\sum_{j=1}^{D-1} \binom{n-k-t}{j}$ such basic memory units for one warp. Together with the result of evaluating \mathcal{S}_i at the current point (which requires one basic unit as well) a warp requires $\sum_{j=0}^{D-1} \binom{n-k-t}{j}$ basic units.

Therefore, in total Gray-code enumeration requires $(2^{t-5} \cdot \sum_{j=0}^{D-1} \binom{n-k-t}{j}) \cdot 32 \cdot (k+1)$ words of size 32-bit in global memory and $\binom{n-k-t}{D}$ words of size 32-bit in constant memory.

3.3 Testing the Solvability of a Linear System

After a linear system has been computed during an iteration step of Gray-code enumeration, the system needs to be checked for solvability. Since the linear system is small, the straight-forward approach for testing its solvability is to simply solve it with Gauss-Jordan elimination.

In the standard Gauss-Jordan elimination algorithm, once a pivot row for the i^{th} pivot element is located, it is moved to its final position by swapping with the i^{th} row. However, we are storing the linear system in column order, so row swapping is expensive. Therefore, we avoid row-swapping by maintaining a mask that tracks which rows are in their final position.

After as many rows as the number of variables k in the linear system \mathcal{S}_i have been marked as final, the algorithm stops. The remaining unmarked rows are redundant equations and their first k coefficients which represent the variables x_1, x_2, \dots, x_k are guaranteed to be zero. Therefore, testing the solvability of \mathcal{S}_i is as simple as checking if the constant term of any of the redundant equations is non-zero.

Clearly, if the system is solvable, a solution can be extracted from the last column based on the first k columns. In particular, the position of 1 in the i^{th} column points to the value for x_i in the last column. Note that before extracting a solution, one has to test whether or not the system is underdetermined. To achieve this, one can simply verify that none of the first k columns is completely zero since one such column implies a missing pivot element. This verification can be done simultaneously while extracting a solution and does not require extra computation.

We avoid storing data for linear system solving in global memory by storing the entire data in registers. In order to make sure that the compiler maps data to registers, we do not use an array data structure to store the data. Instead, we use a Python script to generate unrolled code with distinct variables for all data. However, the consequence of generating CUDA code at compilation time is that the program has to be re-compiled for each choice of k . This takes roughly 6s on an AMD FX-8350 4 GHz processor, which is negligible.

3.4 Probability of False Positives

There are three possible outcomes of solving the linear system: there can be no, one, or more than one solution. The expected outcome is that there is no solution in which case we proceed to the next Gray-code iteration step. Ideally, we find one single solution only once—which then is also a solution for the original quadratic system. However, there is a small probability that a solution for the subsystem \mathcal{S} is not a solution for the original system, i.e., it is a false

positive. Finally, there is also a chance for finding more than one solution which requires further processing.

Suppose we have a random linear system of m equations in \mathbb{F}_2 of n variables. We would like to estimate the probability that this system has at least one solution. Let A be the augmented matrix of this system ($m \times (n + 1)$).

Assume that during Gaussian elimination the upper-left corner is a pivot. This means that there is at least one 1 in the first column ($2^m - 1$ possibilities). The first row with a leading 1 gets swapped to the top, and the rest of the first column is eliminated. There are n other entries in the first row (2^n possibilities). The remaining $(m - 1) \times n$ sub-matrix remain uniformly random.

We can continue this reasoning conclude that if the Gaussian elimination have pivots in columns $a_1 < a_2 < \dots < a_\ell \leq n + 1$ exactly

$$2^{\sum_{j=1}^{\ell} (n+1-a_j)} \left(\prod_{j=1}^{\ell} (2^{m+1-j} - 1) \right)$$

times. Thus, when $m > n$, we can tell that the largest block of consistent systems have pivots $a_1 = 1, a_2 = 2, \dots, a_n = n$, and these number

$$2^{n(n+1)/2} \left(\prod_{j=1}^n (2^{m+1-j} - 1) \right) < 2^{n(n+1)/2} \left(\prod_{j=1}^n (2^{m+1-j}) \right) = 2^{n(m+1)}.$$

There are $2^{m(n+1)}$ possible matrices so probability of a full-rank consistent system is bounded by $2^{-(m-n)}$. More precisely, for large $m = n$, the probability of full-rank consistency is $\frac{1}{2} \cdot \frac{3}{4} \cdot \frac{7}{8} \dots \left(1 - \frac{1}{2^n}\right) \gtrsim p_0 = \prod_{j=1}^{\infty} \left(1 - \frac{1}{2^j}\right) \approx 0.288788$.

In general a full-rank consistent systems occurs with probability roughly

$$2^{-(m-n)} \prod_{j=1}^n (1 - 2^{m-n+j}) \gtrsim p_{m-n} := p_0 \cdot 2^{-(m-n)} / \left(\prod_{j=1}^{m-n} (1 - 2^{-j}) \right).$$

The second largest block of systems (missing a pivot in column n) is less likely by a factor of $\frac{1}{2(2^{(m+1-n)}-1)}$. Systems missing a pivot in column $(n - j)$ are a further factor of $1/2^{j-1}$ less likely. Thus, probability of consistent systems with $(n - 1)$ pivots is $\approx \frac{p_{m-n}}{2(2^{(m+1-n)}-1)} \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^{n-1}}\right) \approx \frac{p_{m-n}}{(2^{(m+1-n)}-1)}$.

The largest block missing two pivots (in columns n and $n - 1$) is a factor $\frac{1}{2^3(2^{(m+1-n)}-1)(2^{(m+2-n)}-1)}$ smaller than full-rank. Each time we move the first missing pivot left there is a factor of $1/2$. Each time we move the second (right-most) missing pivot left there is a factor of $1/4$. Summing over $2^{-i}4^{-j}$ gets a factor of $8/3$, so we end up having probability of missing 2 pivots close to

$$\approx \frac{p_{m-n}}{3(2^{(m+1-n)}-1)(2^{(m+2-n)}-1)}$$

Continuing this argument, we note that the largest term missing k pivots is smaller by a factor of $2^{k(k+1)} \left(\prod_{j=1}^k (2^{m-n+k} - 1) \right)$. Summing over all matrices

missing k pivots, we get a factor of $\lesssim \frac{2}{1} \frac{4}{3} \cdots \frac{2^k}{2^k - 1}$. So the totality of all matrices missing k pivots is $\approx p_{m-n} / \left(\prod_{j=1}^k (2^{m-n+k} - 1) \right) \left(\prod_{j=1}^k (2^k - 1) \right)$.

The probability of a set of consistent equations for large m and n approaches

$$\left(\frac{p_0}{2^{m-n} \left(\prod_{j=1}^{m-n} (1 - 2^{-j}) \right)} \right) \left[\sum_{k=0}^{\infty} \left(\frac{1}{\left(\prod_{j=1}^k (2^{m-n+k} - 1) \right) \left(\prod_{j=1}^k (2^k - 1) \right)} \right) \right].$$

If we only take two terms, it becomes roughly

$$\left(\frac{p_0}{2^{m-n} \left(\prod_{j=1}^{m-n+1} (1 - 2^{-j}) \right)} \right) \rightarrow 2^{-(m-n)} \text{ for large } m - n.$$

This is consistent with intuition. For example, to have no more than 1 consistent system in 1000, we need $m - n \geq 10$. For two further examples, we note that $p_1 = p_0 = 0.288788$. The probability when $m - n = 1$ of a set of consistent equations is approximately

$$p_1 \cdot \left[\sum_{k=0}^{\infty} \left(\frac{1}{\left(\prod_{j=1}^k (2^{k+1} - 1) \right) \left(\prod_{j=1}^k (2^k - 1) \right)} \right) \right] = 0.389678.$$

When $m = n$, the probability of a set of consistent equations is approximately

$$p_0 \sum_{k=0}^{\infty} \frac{1}{\left(\prod_{j=1}^k (2^k - 1) \right)^2} = 0.610322,$$

which is exactly the complement of the previous result!

3.5 Verification of Solution Candidates

When a single solution candidate is found, it needs to be verified with the original \mathcal{MQ} system. Ideally, one would copy the solution candidate from the GPU off-chip memory back to the main memory and verify it on the CPU immediately. In practice, this is not efficient because checking each solution candidate right away on the CPU interrupts the workflow of the GPU. Therefore, an alternative approach is to store all solution candidates in a buffer and only copy them back to the main memory after the GPU kernel finishes. One caveat of this approach is that a sufficiently large buffer must be allocated on the off-chip memory, which may have little capacity left after allocating memory blocks for the data structures used in Gray-code enumeration. If the number of solution candidates is larger than the size of the buffer, some candidates must be dropped.

To avoid this pitfall, we copy some polynomials from the original \mathcal{MQ} system to the GPU which serve as a filter. Evaluating a random polynomial over \mathbb{F}_2 at a random input results in zero with probability 0.5. Therefore, using i polynomials

reduces the number of candidates by a factor of 2^{-i} (for $i \ll n$). Only solution candidates that pass the filter will then be verified with the rest of the equations in the original \mathcal{MQ} system by the CPU.

We are using 32 polynomials that are stored in column-wise format. In this manner, to apply the filter a thread needs to evaluate $\frac{n(n-1)}{2} + n + 1$ monomials in the polynomials with the solution candidate. Therefore, this takes at most $\mathcal{O}(n^2)$ machine instructions. However, filtering only needs to be applied when the linear system has a solution, which happened very rarely during our experiments and its execution time was completely hidden.

If more than one solution is found, more effort is required in order not to miss the solution. The probability of having more than one solution is very small for well chosen implementation parameters (see Sect. 3.4). Therefore, our implementation simply reports when it encounters this case and moves on to the next iteration step. During all our experiments, this case never occurred.

3.6 Pipelining

When external hybridization is applied, i.e. p variables are fixed in the original \mathcal{MQ} system, one has to extract a sub-system and subsequently perform Gray-code enumeration at most 2^p times. Since we perform Gray-code enumeration on the GPU, which operates independently from the CPU, we are able pipeline the two stages. In other words, while performing Gray-code enumeration on the GPU, a sub-system for the next Gray-code enumeration can be computed in parallel on the CPU. In this manner, as long as extracting a sub-system takes at most as much time as Gray-code enumeration, which can be controlled by the choice of p , only the runtime of extracting the first sub-system will manifest.

4 Choice of Parameters

There are several parameters to choose before the Crossbred algorithm can be executed on a CPU/GPU cluster. First, we need to know how many variables k we can keep for linearization. This depends on the Macaulay degree D and the number of variables p fixed in external hybridization. Finally we need to decide how many variables to fix before deploying the workload to the GPUs and how many GPU threads to launch in parallel.

4.1 Number of Variables to Keep

We want to set the parameter k as high as possible in order to reduce the search space for Gray-code enumeration: For every extra variable that can be kept, the search space is halved. As described in the original Crossbred algorithm [11], the maximum of k depends on the Macaulay degree D as well as the number of variables n and the number of equations m in the original \mathcal{MQ} system. The number of linearly independent equations that can be extracted from a Macaulay matrix can be computed as the difference of the number of independent rows

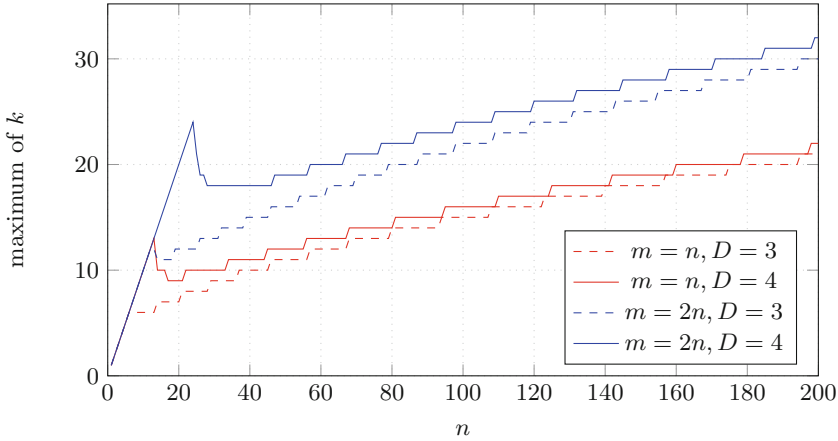


Fig. 1. Maximum number of variables k that can be kept depending on n and m .

$N_{\text{indep_row}}$ in the Macaulay matrix and the number of monomials N_{nl} which are non-linear in x_1, \dots, x_k . This number must be no less than k ; otherwise, there will not be enough equations in the sub-system to obtain a unique solution. The maximum value of k for \mathcal{MQ} systems with $n = m$ and $m = 2n$, based on Macaulay degree $D = 3$ and 4 , can be computed as

$$N_{\text{indep_row}} = \begin{cases} m \cdot (n + 1), & \text{when } D = 3, \\ m \cdot \left(\binom{n}{2} + n + 1 \right) - \left(\binom{m}{2} + m \right), & \text{when } D = 4, \end{cases}$$

$$N_{\text{nl}} = \sum_{i=2}^D \sum_{j=2}^i \binom{k}{j} \cdot \binom{n-k}{i-j},$$

$$N_{\text{indep_row}} - N_{\text{nl}} \stackrel{!}{\geq} k.$$

Figure 1 shows a graph for the number of variables we can keep in relation to the system size for $n < 200$. Clearly, with degree-4 Macaulay matrices one can keep more variables than with degree-3 Macaulay matrices for large enough n . However, for some determined systems, e.g. $n = 140$, using a degree-4 Macaulay matrix does not allow us to keep more variables than when using a degree-3 matrix. In addition, the gap between the two curves for overdetermined systems becomes narrower as n grows. Therefore, similar to determined systems, the effectiveness of degree-4 matrices is expected to become marginal at which point degree-5 Macaulay matrices are required if one wishes to keep considerably more variables than when using degree-3 matrices.

Note that k grows linearly in the beginning of each curve, where the degree of regularity of the \mathcal{MQ} system is smaller than or equal to the Macaulay degree. In this case, a Gröbner basis can be extracted directly from the Macaulay matrix, which immediately yields a solution to the system.

4.2 Macaulay Degree

As discussed in [11], since the Macaulay matrix is used to induce cancellation of the monomials where any of the variables x_1, x_2, \dots, x_k has a degree larger than one, the degree of the Macaulay matrix must be no less than the degree of regularity of a random system of m equations in k variables. In addition, the Macaulay degree is a key factor that determines the maximum value of k . One should therefore choose a Macaulay degree that is larger than the degree of regularity requirement and that can provide a sufficient number of linearly independent equations for the intended value of k .

One caveat of choosing the Macaulay degree is that the memory requirement must be smaller than the available system memory. Since both the number of rows and columns of a Macaulay matrix grow considerably when the degree increases, one might have to choose a smaller Macaulay degree and subsequently a smaller k in case the available memory is insufficient.

Our implementation supports both degree-3 and degree-4 Macaulay matrices. Degree-3 Macaulay matrices are useful for small toy examples, while degree-4 Macaulay matrices are sufficient for the largest problem sizes that we target.

4.3 Number of Variables to Fix During External Hybridization

Section 4.1 gives the formula for computing the maximum value of k for a given system. Since the parameter n in the formula is the number of variables in the \mathcal{MQ} system, one can achieve a higher k by fixing some p variables with external hybridization. In this manner, the number of variables in the system drops by p but the number of equations remains the same. Therefore, the number of variables that can be kept may be higher.

For example, an \mathcal{MQ} system of 148 equations in 74 variables allows to keep $k = 21$ variables with a degree-4 Macaulay matrix. By fixing $p = 4$ variables, it becomes a system in 70 variables, which allows to keep one more variable, i.e., $k = 22$. In this manner, the search space of Gray-code enumeration is split into $2^4 \times 2^{74-4-22}$ instead of $1 \times 2^{74-21}$, which reduces the total number of iterations for Gray-code enumeration by half. On the other hand, 2^p sub-systems of Macaulay matrices need to be computed—so there is a limit on the effectiveness of applying external hybridization.

4.4 Number of Variables to Fix Before Exhaustive Search

In addition to fixing variables by external hybridization, one can further fix some variables in the extracted sub-system *before* entering the exhaustive search stage. By fixing b variables the sub-system beforehand, one can divide the workload evenly into 2^b smaller sub-systems which require less resources for applying exhaustive search. Clearly, since the main purpose of fixing these b variables in the sub-system is to fine-tune the resource requirement, the choice of b should be adjusted based on the hardware architecture and the remaining parameters.

Table 1. Effect of changing the number of GPU threads.

Number of threads	Memory per thread	Total memory	Constant memory	Search space per thread	Runtime (seconds)
2^9	15.41 kB	7.70 MB	5320 B	2^{21}	20.97
2^{10}	14.01 kB	14.01 MB	4560 B	2^{20}	11.93
2^{11}	12.68 kB	25.37 MB	3876 B	2^{19}	7.16
2^{12}	11.42 kB	45.69 MB	3264 B	2^{18}	4.89
2^{13}	10.22 kB	81.81 MB	2720 B	2^{17}	5.15
2^{14}	9.10 kB	145.56 MB	2240 B	2^{16}	5.15
2^{15}	8.04 kB	257.12 MB	1820 B	2^{15}	5.04
2^{16}	7.04 kB	450.50 MB	1456 B	2^{14}	4.95
2^{17}	6.11 kB	782.00 MB	1144 B	2^{13}	4.94
2^{18}	5.25 kB	1343.00 MB	880 B	2^{12}	4.79
2^{19}	4.45 kB	2278.00 MB	660 B	2^{11}	4.86
2^{20}	3.72 kB	3808.00 MB	480 B	2^{10}	4.86

4.5 Number of GPU Threads

Typically, more threads than available cores are launched on a GPU in order to hide memory and instruction latencies. Therefore, there should be a certain threshold for the number of threads after which increasing the number of threads on the GPU does not have an influence on the performance anymore. To find this threshold, we performed a series of experiments by running our GPU kernel on a randomly generated \mathcal{MQ} system of 92 equations in 46 variables with different numbers of 2^t threads. We performed the experiments on a Nvidia Quadro M1000M GPU using the following settings:

- GPU: Nvidia Quadro M1000M, 4 GB off-chip memory, 512 CUDA cores
- Macaulay degree: $D = 3$
- external hybridization: $p = 0$
- Number of variables to fix before enumeration: $b = 0$
- Number of variables to keep: $k = 16$

The results are given in Table 1. As expected, the runtime basically remains constant for $t \geq 12$. For $t < 12$, the degree of parallelism is not sufficient and the latencies manifest.

When $t = 9$, there are $2^9 = 512$ GPU threads deployed, which is exactly the number of CUDA cores available on this particular GPU. In this case, the workload is evenly distributed to all the CUDA cores. Nevertheless, the degree of parallelism is far from enough because executing one single thread per CUDA core is not enough to hide latencies. For example, when the thread loads data from the global memory, which requires hundreds of cycles to access, there is no other thread that can take over the execution resources. Therefore, the CUDA core has no choice but to stall.

Starting from $t = 10$, there are several threads per CUDA core and some latencies can be hidden. The performance gradually improves until $t = 12$, where the degree of parallelism reaches a point where deploying more threads does not improve the ability of the GPU to hide latencies anymore. Therefore, for these experimental settings the threshold where the optimal performance of our implementation can be achieved is 2^{12} .

Note that as explained in Sect. 3.2, for each doubling in the number of GPU threads the amount of global memory required for a single GPU thread reduces slightly but the total amount of memory required for the GPU kernel increases nearly twofold. However, since the last-level derivatives stored in constant memory are shared by all threads, constant memory requirement decreases as t increases.

5 Evaluation

We evaluated the performance of our implementation on the *Saber clusters* [2]. Saber is located at Eindhoven University of Technology and Saber2 at University of Illinois at Chicago. The clusters consist of mostly homogeneous workstations. Out of all the nodes in these two clusters, we used 27 cluster nodes, each equipped with two Nvidia graphics cards. Twelve out of those 27 nodes have two GTX 780 graphics cards while the remaining 15 nodes have two GTX 980 cards. Each node has 32 GB RAM and one AMD FX-8350 4 GHz processor, which has four physical CPU *modules* (similar to a physical core in an Intel CPU) shared by eight logical threads (similar to Intel’s hyper threading), 16 kB L1 data cache per thread, 2 MB L2 cache per module, and 8 MB L3 cache shared by the whole CPU. We used CUDA version 7.5 and compiled our implementation with the back-end compiler bundled with CUDA, which is GCC version 4.8.

We compare our results to the FES implementations on GPUs from [3] and on FPGAs from [4] and to the Crossbred implementation on CPUs from [11]. Since [3] is using an older GTX 295 graphics card, we scale their results as follows: The GTX 295 graphics card has 480 CUDA cores running at 1242 MHz. Our GTX 980 graphics card has 2048 CUDA cores running at 1278.50 MHz. Therefore, we scale the results of [3] by a factor of $\frac{1242}{1278} \cdot \frac{480}{2048}$ in order to achieve a rough comparison of the performance. This over-estimates the power of a GTX 295 compared to a GTX 780 and therefore is in favor of [3] in some of the comparisons.

5.1 Overdetermined Systems—Fukuoka \mathcal{MQ} Challenge

We solved some of the *Fukuoka \mathcal{MQ} challenges* using our implementation. These challenges were created in 2015 in order to help determining appropriate parameters for public-key cryptographic schemes based on \mathcal{MQ} systems. In particular, we chose to target Type-I challenges generated with seed 4 because they consist of \mathcal{MQ} systems in n variables and $m = 2n$ equations over \mathbb{F}_2 .

The experimental results of solving Type-I challenges for $n \in \{55, \dots, 67\}$ using *one single* GTX 980 graphics card are given in Table 2. The workflow

Table 2. Solving overdetermined systems with a single GTX 980 graphics card.

n	Parameters (D, p, k, b, t)	Search space $2^p \times 2^b \times 2^{n-p-k-b}$	Extracting sub-systems (seconds)	Exhaustive search (seconds)	Total runtime (seconds)	Worst-case runtime (seconds)
55	(4, 0, 19, 0, 14)	$1 \times 1 \times 2^{36}$	387.80	318.25	706.20	706.20
56	(4, 1, 20, 0, 14)	$2^1 \times 1 \times 2^{35}$	491.60 (1)	169.94 (1)	658.67	1317.34
57	(4, 0, 20, 0, 14)	$1 \times 1 \times 2^{37}$	606.75	650.90	1258.73	1258.73
58	(4, 0, 20, 0, 14)	$1 \times 1 \times 2^{38}$	670.26	1311.97	1982.74	1982.74
59	(4, 0, 20, 0, 14)	$1 \times 1 \times 2^{39}$	741.62	2619.00	3361.77	3361.77
60	(4, 0, 20, 0, 14)	$1 \times 1 \times 2^{40}$	782.12	5211.05	5994.41	5994.41
61	(4, 0, 20, 1, 14)	$1 \times 2^1 \times 2^{40}$	872.34	5204.18 (1)	6077.13	11280.34
62	(4, 0, 20, 2, 14)	$1 \times 2^2 \times 2^{40}$	920.24	10485.95 (2)	11407.64	21892.14
63	(4, 4, 21, 0, 14)	$2^4 \times 1 \times 2^{38}$	9406.21 (11)	14827.94 (11)	24234.15	35250.72
64	(4, 3, 21, 1, 13)	$2^3 \times 2^1 \times 2^{39}$	1991.48 (2)	10469.58 (4)	12456.97	49844.24
65	(4, 3, 21, 2, 14)	$2^3 \times 2^2 \times 2^{39}$	1046.62 (1)	10517.21 (4)	11565.10	92510.64
66*	(4, 1, 21, 5, 13)	$2^1 \times 2^5 \times 2^{39}$	16268.10 (2)	133896.93 (51)	151867.70	184295.62
67*	(4, 0, 21, 7, 13)	$1 \times 2^7 \times 2^{39}$	10298.95	198835.78 (74)	209172.34	354231.11

Table 3. Solving overdetermined systems using 27 nodes of the Saber clusters.

n	Parameters (D, p, k, b, t)	Search space $2^p \times 2^b \times 2^{n-p-k-b}$	Extracting sub-systems (seconds)	Total runtime (seconds)	Worst-case runtime (GPU-hours)
68	(4, 6, 21, 2, 13)	$2^6 \times 2^2 \times 2^{39}$	9799.15	12802.11	214.45
69	(4, 8, 22, 0, 13)	$2^8 \times 1 \times 2^{39}$	11238.49	56697.70	229.10
70	(4, 7, 22, 2, 13)	$2^7 \times 2^2 \times 2^{39}$	14367.71	44223.81	452.65
71	(4, 8, 22, 2, 13)	$2^8 \times 2^2 \times 2^{39}$	14392.00	87415.91	947.20
72	(4, 9, 22, 2, 13)	$2^9 \times 2^2 \times 2^{39}$	13912.39	144145.58	1867.44
73	(4, 8, 22, 4, 13)	$2^8 \times 2^4 \times 2^{39}$	18055.07	159585.32	3700.87
74	(4, 10, 22, 3, 13)	$2^{10} \times 2^3 \times 2^{39}$	15163.72	118323.38	8236.05

of the algorithm, i.e., how the search space is split and enumerated, is listed in the 3rd column of the table. For parameters $p > 0$ and $b > 0$, external hybridization and Gray-code enumeration need to be repeated at most 2^p and 2^b times respectively. The numbers inside the parentheses in the 4th and 5th column specify how many repetitions were performed during the experiments. For all these small experiments we used the GPU instead of the CPU to extract sub-systems except for the last two experiments marked with an asterisk, because the reduced Macaulay matrix was too large to fit into the 4GB off-chip memory of the GTX 980 graphics card. As shown in Table 2, solving an \mathcal{MQ} system of 134 equations in 67 variables requires at most 354231.11 s which equals to 98.39 h on a single GPU, including the computation time of extracting sub-systems.

For larger Type-I challenges with $n \in \{68, \dots, 74\}$ we used 27 nodes in the Saber and Saber2 clusters by distributing the 2^p smaller \mathcal{MQ} systems obtained from external hybridization evenly over the nodes. The results are given in Table 3, which basically has the same format and notation as Table 2. In these larger experiments, sub-systems were extracted from degree-4 Macaulay matrices with the CPU because the GPU off-chip memory cannot accommodate the size of

the reduced Macaulay matrices. However, these parameters allowed us to pipeline the extraction of sub-systems on the CPU and the exhaustive search stage on the GPU. Therefore, the computation time of the former can be completely hidden except in the first run. Some of the cluster nodes we used have GTX 780 graphics cards with only 3 GB of off-chip memory while GTX 980 graphics cards have 4 GB. Therefore, we adjusted the parameters t and b according to the memory size of the GTX 780. Consequently, the 4 GB off-chip memory on the GTX 980 was not fully utilized but there was no noticeable impact on performance.

Impact of k . The experiments show that despite the number of variables increasing by one for each experiment, whenever the maximum value of the parameter k increases (either with or without external hybridization), the runtime almost stays the same. For example, for the overdetermined \mathcal{MQ} system \mathcal{F}_{68} , $n = 68$ by keeping $k = 21$ variables, there are 47 variables left in \mathcal{F}_{68} to enumerate (see Table 3). For the overdetermined \mathcal{MQ} system \mathcal{F}_{69} , $n = 69$, the maximum value of k can be increased by one (with external hybridization), so $k = 22$ variables can be kept. Therefore, there are also 47 variables to enumerate for \mathcal{F}_{69} . Hence, for both systems the total maximum number of iterations that need to be performed during Gray-code enumeration is the same. However, since for \mathcal{F}_{69} linear systems in 22 variables instead of 21 have to be computed, the cost of each iteration of Gray-code enumeration for \mathcal{F}_{69} is slightly larger than for \mathcal{F}_{68} . Thus, the worst-case runtime for $n = 69$ is slightly larger than for $n = 68$.

Comparison. Previous records of solving Type-I challenges were held by the FES and Crossbred algorithms. The FES implementation for FPGAs is able to perform full enumeration over the search space for an \mathcal{MQ} system in 64 variables in 956 days [4]. Therefore, it solves a \mathcal{MQ} system of 148 equations in 74 variables in at most $2^{74-64} \cdot 956$ days ≈ 2900 FPGA-years. The corresponding GPU implementation in [3] requires 21 min to solve an \mathcal{MQ} system with $n = 48$ variables on a GTX 295 graphics card. Scaling the performance on the GTX 295 to our graphics cards as described before results in $2^{74-48} \cdot 21 \text{ min} \cdot \frac{1242}{1287} \cdot \frac{480}{2048} \approx 610$ GPU-years. The original Crossbred implementation for CPUs requires at most 41 CPU-years to solve the challenge [11] using $k = 23$. As shown in Table 3, our implementation is most efficient with $k = 22$ and requires at most 8236 GPU-hours, i.e., 0.94 GPU-years. Table 4 shows an overview of the comparison including the respective speedup of our implementation.

Estimated Security for $n = 74$, $m = 2n$. As mentioned before, a GTX 980 graphics card consists of 2048 CUDA cores operating at 1278.50 MHz. Based on profiling information, our implementation achieves 37% GPU utilization. Therefore, we estimate the security strength of this particular \mathcal{MQ} system, defined as the number of operations required to solve the system, as

$$8236.05 \cdot 2048 \cdot 1278.50 \cdot 10^6 \cdot 3600 \cdot 0.37 \approx 2^{64.6}.$$

Thus, an \mathcal{MQ} system with $n = 74$, $m = 2n$ only provides about 64-bit security.

Table 4. Worst-case runtime and speedup of our work compared to previous work.

n	m	k	Approach	Worst-case runtime	Our speedup
74	148	–	[4] (2014)	2900 FPGA-years ^a	3100.0
74	148	–	[3] (2010)	610 GPU-years ^{a,b}	650.0
74	148	23	[11] (2017)	41 CPU-years	44.0
74	148	22	Our (2018)	0.94 GPU-years	1.0
46	46	12	[11] (2017)	1900 CPU-seconds	23.0
46	46	12	Our (2018)	82 GPU-seconds	1.0
59	59	–	[4] (2014)	30 FPGA-days ^a	12.0
59	59	–	[3] (2010)	6.8 GPU-days ^{a,b}	2.8
59	59	13	Our (2018)	2.4 GPU-days	1.0

^aextrapolated; ^bscaled from GTX 259 to GTX 980

5.2 Determined Systems

Determined systems with $n = m$ are not included in the Fukuoka \mathcal{MQ} challenges. Therefore, we performed experiments for such systems using randomly generated, solvable systems. We solved those systems with one single GTX 980 graphics card on a node in the Saber2 cluster. The experimental results are given in Table 5, whose format and notation is the same as Table 2.

For determined systems, the number of variables that can be kept is much smaller than for overdetermined systems due to that fact that fewer equations are available. However, the linear systems that are enumerated during Gray-code enumeration consist of fewer variables. Therefore, the cost of each iteration is also lower. As Table 5 shows, solving a determined \mathcal{MQ} system in n variables is roughly as difficult as solving an overdetermined \mathcal{MQ} system where $m' = 2n'$, $n' = n+7 \sim n+8$. Nevertheless, Fig. 1 shows that the gap between the number of variables that can be kept for determined and overdetermined systems gradually becomes larger as n grows. Therefore, this observation only applies to the systems in Table 5 but not to larger determined systems, e.g. $n = 172$.

Comparison. The extrapolated worst-case runtime of the FES algorithm on FPGAs from [4] is $2^{59-64} \cdot 956$ days ≈ 30 FPGA-days. The corresponding runtime on GPUs [3] is $2^{59-48} \cdot 21$ min $\cdot \frac{1242}{1278} \cdot \frac{480}{2048} \approx 6.8$ GPU-days. Our implementation requires at most 210601 s, i.e., about 2.4 GPU-days. Table 4 shows the speedup of our implementation. Our speedup over FES for $n = m = 59$ is significantly lower than for $n = 74, m = 148$. This shows that the Crossbred algorithm is less efficient for small k and therefore more suitable for larger systems and for overdetermined systems. The authors of the Crossbred-CPU implementation in [11] do not provide performance numbers for $n = m = 59$. Therefore, we show a comparison for $n = m = 46$ in Table 4.

Table 5. Solving determined systems with a single GTX 980 graphics card.

n	Parameters (D, p, k, b, t)	Search space $2^p \times 2^b \times 2^{n-p-k-b}$	Extracting sub-systems (seconds)	Exhaustive search (seconds)	Total runtime (seconds)	Worst-case runtime (seconds)
46	(4, 0, 12, 0, 16)	$1 \times 1 \times 2^{34}$	33.90	47.63	82.12	82.12
47	(4, 0, 12, 0, 15)	$1 \times 1 \times 2^{35}$	36.31	96.12	132.92	132.92
48	(4, 0, 12, 0, 15)	$1 \times 1 \times 2^{36}$	39.76	190.59	230.88	230.88
49	(4, 0, 12, 0, 15)	$1 \times 1 \times 2^{37}$	42.98	380.91	424.48	424.48
50	(4, 0, 12, 0, 15)	$1 \times 1 \times 2^{38}$	46.86	754.86	802.34	802.34
51	(4, 0, 12, 0, 15)	$1 \times 1 \times 2^{39}$	50.74	1542.07	1593.46	1593.46
52	(4, 0, 12, 0, 14)	$1 \times 1 \times 2^{40}$	53.59	3049.07	3103.21	3103.21
53	(4, 0, 12, 1, 14)	$1 \times 2^1 \times 2^{40}$	57.05	6249.61 (2)	6307.22	6307.22
54	(4, 0, 12, 2, 14)	$1 \times 2^2 \times 2^{40}$	60.86	3141.67 (1)	3205.11	12635.54
55	(4, 1, 13, 1, 14)	$2^1 \times 2^1 \times 2^{40}$	95.30 (1)	3322.54 (1)	3418.48	13480.76
56	(4, 0, 13, 3, 14)	$1 \times 2^3 \times 2^{40}$	118.85	6600.55 (2)	6720.12	26521.05
57	(4, 0, 13, 4, 14)	$1 \times 2^4 \times 2^{40}$	121.54	46053.43 (14)	46175.72	52754.03
58	(4, 0, 13, 5, 14)	$1 \times 2^5 \times 2^{40}$	133.97	105432.90 (32)	105567.66	105567.66
59	(4, 0, 13, 6, 14)	$1 \times 2^6 \times 2^{40}$	144.13	197303.32 (60)	197448.24	210601.00

“80-bit Security”. Sakumoto et al. propose an \mathcal{MQ} -based public-key identification schemes and “80-bit secure” parameters $n = 84, m = 80$ in [15]. When using our implementation and hardware for solving such systems, the best configuration is $(D, p, k, b, t) = (4, 27, 16, 1, 14)$, because this choice gives the largest k for the smallest p such that the computation on one Macaulay matrix does not take more time than the corresponding computations on the GPU. Therefore, the runtime of extracting the sub-systems can be completely hidden by pipelining CPU and GPU computations. Extracting a sub-system with these parameters takes 985.86 s and each GPU kernel launch takes on average 4338.59 s for 2^{40} iterations. The worst-case runtime for solving the \mathcal{MQ} system is therefore $4338.59 \text{ s} \cdot 2^{(84-16-40)} \approx 37000$ GPU-years.

However, since the probability of obtaining a solution for a determined system is approximately $1 - \frac{1}{e} \approx 0.63$ [9] and the runtime r for exploring a sub-space of size 2^{80-16} is $r = 4338.59 \text{ s} \cdot 2^{(80-16-40)} \approx 2300$ GPU-years (with the parameters as above), the expected runtime of solving an \mathcal{MQ} system where $n = 84, m = 80$ is only

$$r \cdot \left(1 - \frac{1}{e}\right) \cdot \sum_{i=1}^{\infty} i \frac{1}{e^{i-1}} = r \cdot \frac{e}{e-1} \approx 3600 \text{ GPU-years.}$$

Following the calculation in Sect. 5.1, the expected number of operations required for solving such a system on a GPU is therefore

$$3600 \cdot 2048 \cdot 1278.50 \cdot 10^6 \cdot 365 \cdot 24 \cdot 3600 \cdot 0.3706 \approx 2^{76.5},$$

i.e., these parameters are roughly “76-bit secure” which is very close to the security claimed in [15]. Due to the small k , the Crossbred algorithm gives only a moderate improvement over the FES algorithm as in [3] with an expected cost of roughly $2^{80} \cdot 4 \cdot \frac{e}{e-1} \approx 2^{82.7}$ GPU-operations.

However, solving the underlying \mathcal{MQ} systems of this public-key identification scheme using the security parameters of [15] is feasible on average within about

one year using 3600 GTX 980 graphics cards at the cost of electricity and about \$2 million US dollars for hardware, assuming a price of \$550 US dollars per GTX 980 graphics card². This shows that breaking 80-bit security is within reach at moderate cost and time using today's technology and that 128-bit security must be the minimum requirement for multivariate cryptography.

Acknowledgments. We would like to thank Daniel J. Bernstein for granting us access to his Saber GPU clusters at Eindhoven University of Technology and the University of Illinois at Chicago. This research was partially supported by the project MOST105-2923-E-001-003-MY3 of the Ministry of Science and Technology, Taiwan.

References

1. Berbain, C., Gilbert, H., Patarin, J.: QUAD: a practical stream cipher with provable security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 109–128. Springer, Heidelberg (2006)
2. Bernstein, D.J.: The saber cluster. <https://blog.cr.yyp.to/20140602-saber.html>
3. 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)
4. Bouillaguet, C., Cheng, C.-M., Chou, T., Niederhagen, R., Yang, B.-Y.: Fast exhaustive search for quadratic systems in \mathbb{F}_2 on FPGAs. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 205–222. Springer, Heidelberg (2014)
5. Clough, C., Baena, J., Ding, J., Yang, B.-Y., Chen, M.: Square, a new multivariate encryption scheme. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 252–264. Springer, Heidelberg (2009)
6. Ding, J., Schmidt, D.: Rainbow, a new multivariable polynomial signature scheme. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 164–175. Springer, Heidelberg (2005)
7. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F_4). *J. Pure Appl. Algebra* **139**, 61–88 (1999)
8. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In: International Symposium on Symbolic and Algebraic Computation – ISSAC 2002, pp. 75–83. ACM Press (2002)
9. Fusco, G., Bach, E.: Phase transition of multivariate polynomial systems. *Math. Struct. Comput. Sci.* **19**, 9–23 (2009)
10. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996)
11. Joux, A., Vitse, V.: A crossbred algorithm for solving boolean polynomial systems. *IACR Cryptology ePrint Archive* (2017). <https://eprint.iacr.org/2017/372>
12. 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)

² <https://www.anandtech.com/show/8526/nvidia-geforce-gtx-980-review>.

13. Patarin, J., Courtois, N., Goubin, L.: QUARTZ, 128-bit long digital signatures. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 282–297. Springer, Heidelberg (2001)
14. Porras, J., Baena, J., Ding, J.: ZHFE, a new multivariate public key encryption scheme. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 229–245. Springer, Cham (2014)
15. Sakumoto, K., Shirai, T., Hiwatari, H.: Public-key identification schemes based on multivariate quadratic polynomials. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 706–723. Springer, Heidelberg (2011)
16. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Foundations of Computer Science, pp. 124–134. IEEE (1994)
17. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **41**(2), 303–332 (1999)
18. Szepieniec, A., Ding, J., Preneel, B.: Extension field cancellation: a new central trapdoor for multivariate quadratic systems. In: Takagi, T. (ed.) PQCrypto 2016. LNCS, vol. 9606, pp. 182–196. Springer, Cham (2016)
19. Yang, B.-Y., Chen, J.-M.: All in the XL family: theory and practice. In: Park, C., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 67–86. Springer, Heidelberg (2005)
20. Yang, B.-Y., Chen, O.C.-H., Bernstein, D.J., Chen, J.-M.: Analysis of QUAD. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 290–308. Springer, Heidelberg (2007)



Practical Cryptanalysis of a Public-Key Encryption Scheme Based on Non-linear Indeterminate Equations at SAC 2017

Keita Xagawa^(✉)

NTT Secure Platform Laboratories, 3-9-11, Midori-cho,
Musashino-shi, Tokyo 180-8585, Japan
xagawa.keita@lab.ntt.co.jp

Abstract. We investigate the security of a public-key encryption scheme, the Indeterminate Equation Cryptosystem (IEC), introduced by Akiyama, Goto, Okumura, Takagi, Nuida, and Hanaoka at SAC 2017 as post-quantum cryptography. They gave two parameter sets PS1 $(n, p, \deg X, q) = (80, 3, 1, 921601)$ and PS2 $(n, p, \deg X, q) = (80, 3, 2, 58982400019)$.

The paper gives practical key-recovery and message-recovery attacks against those parameter sets of IEC through lattice basis-reduction algorithms. We exploit the fact that $n = 80$ is composite and adopt the idea of Gentry's attack against NTRU-Composite (EUROCRYPT2001) to this setting. The summary of our attacks follows:

- On PS1, we recover 84 private keys from 100 public keys in 30–40 s per key.
- On PS1, we recover partial information of all message from 100 ciphertexts in a second per ciphertext.
- On PS2, we recover partial information of all message from 100 ciphertexts in 30 s per ciphertext.

Moreover, we also give message-recovery and distinguishing attacks against the parameter sets with prime n , say, $n = 83$. We exploit another subring to reduce the dimension of lattices in our lattice-based attacks and our attack succeeds in the case of $\deg X = 2$.

- For PS2' $(n, p, \deg X, q) = (83, 3, 2, 68339982247)$, we recover 7 messages from 10 random ciphertexts within 61,000 s ≈ 17 h per ciphertext.
- Even for larger n , we can find short vectors from lattices to break the underlying assumption of IEC. In our experiment, we can find such vector within 330,000 s ≈ 4 days for $n = 113$.

Keywords: Public-key encryption · Indeterminate Equations Cryptosystem · Post-quantum cryptography

1 Introduction

Algebraic-Surface Cryptosystem (ASC) is a public-key cryptosystem based on the section-finding problem [AG06, AGM09]. Recently, the new version

of ASC, the IEC encryption scheme, was proposed by Akiyama et al. at SAC 2017 [AGO+18], where IEC stands for Indeterminate Equation Cryptosystem. Let $R_{n,q} := \mathbb{Z}_q[t]/(t^n - 1)$ and consider $R_{n,q}[x, y]$. The section-finding problem over $R_{n,q}[x, y]$ is, given an algebraic surface $X(x, y) = 0$, finding the section $u = (u_x, u_y) \in R_{n,q}^2$ such that $X(u_x, u_y) = 0$ [AGO+18]. The authors investigate the security of IECs by considering the lattice-based attacks and define two sets of parameter values, PS1 $(n, p, \deg X, q) = (80, 3, 1, 921601)$ and PS2 $(n, p, \deg X, q) = (80, 3, 2, 58982400019)$.

1.1 Our Contribution

We give practical-time lattice-based attacks against the IECs.

Our first attack is combining the original lattice-based attack with Gentry’s attack [Gen01] against NTRU Composite [Sil01]. Let d be a non-trivial divisor of n , say, 40. We can consider the subring $R_{d,q}[x, y]$ instead of $R_{n,q}[x, y]$. This modification allows us to employ a smaller lattice than that in the original lattice-based attacks. Our attack succeeds as follows:

- On PS1, we mount a key-recovery attack. Our attack finds 84 secret keys from 100 random keys. The attack takes approximately 30 s per key.
- On PS1, we mount a partial-message-recovery attack. Our attack finds partial messages of all 100 pairs of random public key and ciphertext. The attack takes approximately 0.5 s per try.
- On PS2, we mount a partial-message-recovery attack. Our attack finds partial messages of all 100 pairs of random public key and ciphertext. The attack takes approximately 30 s per try.

We exploit another class of subring $R_{n,q}[x]$ of $R_{n,q}[x, y]$ to reduce the dimension of lattices in our lattice-based attacks. Our attack succeeds in the case of $\deg X = 2$ as follows:

- For $(n, p, \deg X) = (83, 3, 2)$, we recover 7 messages out of 10 random ciphertexts in 61,000 s \approx 17 h per ciphertext.
- Even for larger n , we can find short vector which enables us to break the underlying assumption of IEC. We can find such vector for $n = 113$ within 330,000 s \approx 4 days.

Responsible Disclosure Process: We already notified the authors of our attacks before making this paper public. We informed them by email on September 28th with key-recovery attack on PS1, October 2nd with partial-message-recovery attack on PS1 and PS2, October 17th with message-recovery attack on $(n, p, \deg X) = (83, 3, 2)$, and November 2nd with distinguishing attack on variant of PS2 with $n \geq 83$. The authors reported that they have changed parameter values and they run their experiments further. We publish this paper after Akiyama et al. published their revised paper and their NIST PQC submission [AGO+17b, AGO+17a].

1.2 Organization

We define notations and review lattices in Sect. 2. We review the IEC scheme in Sect. 3 and the original lattice-based attacks in Sect. 4. We recall Gentry's attack in Sect. 5. We combine them in Sect. 6. We also give new attacks in Sect. 7. The experimental results are reported in Sect. 8.

2 Preliminaries

Notations: The security parameter is denoted by κ .

For a positive integer q , we define $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$ and $\mathbb{Z}_q^+ := \{0, 1, \dots, q-1\}$. For a positive integer n , we define $R_n := \mathbb{Z}[t]/(t^n - 1)$. For two positive integers n and q , we define $R_{n,q} := \mathbb{Z}_q[t]/(t^n - 1)$. We also define a subset $R_{n,q,p}$ of $R_{n,q}$ as a set of all \mathbb{Z}_p^+ -coefficient polynomials in $R_{n,q}$, that is,

$$R_{n,q,p} := \left\{ f = \sum_{i=0}^{n-1} f_i t^i \in R_{n,q} \mid f_i \in \{0, 1, \dots, p-1\} \subset \mathbb{Z}_q \right\}.$$

Let R be a ring and consider $R[x, y]$. For R and a set of indices $\Gamma \subseteq \mathbb{Z}_{\geq 0}^2$, we define

$$\mathfrak{F}(\Gamma, R) := \left\{ f \in R[x, y] \mid f = \sum_{(i,j) \in \Gamma} a_{ij} x^i y^j \right\},$$

a set of all polynomials in $R[x, y]$ which only consists of $x^i y^j$ terms for $(i, j) \in \Gamma$. We will refer Γ as the term set. (Those notations are borrowed from [AGO+18].) We define *the total degree* of $f(x, y) \in R[x, y]$ as the maximum of the sums of the exponents of the variables in the term $a_{ij} x^i y^j$.

Polynomials: We review the notations which bridge polynomials in R_n and n -dimensional vectors (and matrices). For integers n and q , let us define two functions:

$$\begin{aligned} \text{vec}_n : R_{n,q} &\rightarrow \mathbb{Z}^n : f = f_0 + f_1 t + \dots + f_{n-1} t^{n-1} \mapsto (f_0, f_1, \dots, f_{n-1}) \\ \text{Rot}_n : R_{n,q} &\rightarrow \mathbb{Z}^{n \times n} : f \mapsto \{f_{j-i \bmod n}\}_{i,j=0,\dots,n-1} = \begin{pmatrix} \text{vec}_n(f) \\ \text{vec}_n(tf) \\ \text{vec}_n(t^2 f) \\ \vdots \\ \text{vec}_n(t^{n-1} f) \end{pmatrix}. \end{aligned}$$

We have

$$\text{vec}_n(f) \cdot \text{Rot}_n(g) = \text{vec}_n(f \cdot g) \text{ and } \text{Rot}_n(f) \cdot \text{Rot}_n(g) = \text{Rot}_n(f \cdot g)$$

Lattices: Given n -linearly independent vectors $B = \{b_0, \dots, b_{n-1}\} \subset \mathbb{R}^m$, the lattice generated by them is the set of vectors

$$\mathcal{L}(B) = \mathbb{Z}^n \cdot B = \left\{ \sum_{i=0}^{n-1} x_i b_i \mid x_i \in \mathbb{Z} \right\}.$$

The vectors B are known as a basis of the lattice. If $n = m$, we say the lattice is the full-rank. In what follows, we only consider full-rank lattices.

The determinant or volume $\text{vol}(\Lambda)$ of a full-rank lattice Λ is the absolute value of the determinant of any given basis B of Λ , that is, $\text{vol}(\Lambda) = |\det(B)|$. The dual of a lattice Λ , denoted by Λ^* , is the lattice consisting of the set of all vectors $z \in \mathbb{R}^m$ orthogonal to any vectors $v \in \Lambda$, that is, $\Lambda^* = \{z \in \mathbb{R}^m \mid \langle z, y \rangle = 0 \text{ for all } y \in \Lambda\}$.

We also define q -ary lattices. For $A \in \mathbb{Z}_q^{n \times m}$,

$$\begin{aligned} \Lambda_q(A) &:= \{z \in \mathbb{Z}^m \mid z = sA \pmod{q} \text{ for some } s \in \mathbb{Z}^n\} \\ \Lambda_q^\perp(A) &:= \{e \in \mathbb{Z}^m \mid eA^\top \equiv 0 \pmod{q}\}. \end{aligned}$$

We have

$$\Lambda_q^\perp(A) = q \cdot \Lambda_q(A)^* \text{ and } \Lambda_q(A) = q \cdot \Lambda_q^\perp(A)^*.$$

See e.g., [GPV08, Sect. 5].

The basis of Λ_q is easily obtained. For example, we obtain the basis by considering a matrix $\begin{pmatrix} A \\ qI_m \end{pmatrix}$ and taking the row echelon form of the matrix.

SVP and CVP: Finally we define shortest-vector problem and closest-vector problem. The shortest-vector problem (SVP) is, given a lattice Λ , finding a non-zero vector $v \in \Lambda \setminus \{0\}$ such that $\|v\| \leq \|x\|$ for any non-zero lattice vector $x \in \Lambda \setminus \{0\}$. The closet-vector problem (CVP) is, given a lattice Λ and a target vector t , finding a lattice vector $w \in \Lambda$ such that $\|w - t\| \leq \|x - t\|$ for any lattice vector $x \in \Lambda$.

The Gaussian heuristic says that the m -dimensional full-rank lattice contains a short vector of length approximately

$$\gamma = \sqrt{\frac{m}{2\pi e}} \det(L)^{1/m}.$$

If our target vector v is sufficiently smaller than γ , then we expect the LLL or BKZ algorithm find the short vector v .

3 IEC Scheme

Parameters: In the IEC scheme, we will employ $X \in R[x, y]$ as a public key, $r, e \in R[x, y]$ as a random polynomials in ciphertexts. The IEC involves several parameters, (p, q, n) and $(\Gamma_X, \Gamma_r, \Gamma_{Xr})$:

1. p, q : primes and $p \ll q$
2. n : the degree of $R_{n,q} = \mathbb{Z}_q[t]/(t^n - 1)$
3. Γ_X : The term set of $X(x, y)$

- 4. w_X : The total degree of X
- 5. Γ_r : The term set of the random polynomial $r(x, y)$
- 6. w_r : The total degree of r
- 7. Γ_{Xr} : The term set of the random polynomial $e(x, y)$

Akiyama et al. defined

$$\Gamma_{Xr} := \{(i, j) + (k, l) \mid (i, j) \in \Gamma_X, (k, l) \in \Gamma_r\}$$

in order to avoid the linear algebraic attacks against the previous cryptosystems [AGO+18, Sect. 2.2]. They also require large q as

$$q > \#\Gamma_{Xr} \cdot p(p-1) \cdot (n(p-1))^{w_X+w_r} \tag{1}$$

to make the scheme perfectly correct. They implicitly defined

$$\Gamma_X = \{(i, j) \in \mathbb{Z}_{\geq 0}^2 \mid i + j \leq w_X\} \text{ and } \Gamma_r = \{(i, j) \in \mathbb{Z}_{\geq 0}^2 \mid i + j \leq w_r\}.$$

Although Γ_X and Γ_r can be different, they always take $\Gamma_X = \Gamma_r$. Hence, they just parameterize $\deg X$ instead of w_X and w_r . They give two sets of parameter values in Table 1.

Table 1. Proposed sets of parameter values [AGO+18, Table 3]. PS2' is obtained by setting $n = 83$ in PS2.

	n	p	q	$\deg X$	$\deg r$	$\#\Gamma_{Xr}$	$ sk $ (bits)	$ pk $ (bits)	$ ct $ (bits)
PS1	80	3	921601	1	1	6	256	4755	9510
PS2	80	3	58982400019	2	2	15	256	17174	42935
PS2'	83	3	68339982247	2	2	15	264	17928	44820

Key Generation: The secret key is a *small* solution of the indeterminate equation $X(x, y) = 0$. We denote the solution by

$$u : (x, y) = (u_x(t), u_y(t)) \in R_{n,q,p}^2.$$

The public key is the indeterminate equation $X(x, y) = 0$ that has a small solution u . We denote it by

$$X(x, y) = \sum_{(i,j) \in \Gamma_X} a_{ij} x^i y^j, \text{ where } a_{ij} \in R_{n,q}.$$

Akiyama et al. recommend to choose a_{ij} except a_{00} uniformly at random and set $a_{00} := -\sum_{(i,j) \in \Gamma_X \setminus \{(0,0)\}} a_{ij} u_x^i u_y^j$.

Encryption: A plaintext is treated as $m(t) \in R_{n,q,p}$. The ciphertext is

$$c(x, y) := m(t) + X(x, y) \cdot r(x, y) + p \cdot e(x, y) \in \mathfrak{F}(\Gamma_{Xr}, R_{n,q}),$$

where we choose $r(x, y) \leftarrow \mathfrak{F}(\Gamma_r, R_{n,q})$ and $e(x, y) \leftarrow \mathfrak{F}(\Gamma_{Xr}, R_{n,q,p})$.

Decryption: Given a ciphertext $c(x, y) \in \mathfrak{F}(\Gamma_{Xr}, R_{n,q})$,

1. compute $c(u_x, u_y) \in R_{n,q}$
2. regard $c(u_x, u_y)$ as a polynomial in $R_n (= \mathbb{Z}[t]/(t^n - 1))$, compute $m'(t) := c(u_x, u_y) \bmod p$, and output $m'(t)$

Notice that $c(u_x, u_y) = m(t) + p \cdot e(u_x, u_y) \in R_{n,q}$ because $X(u_x, u_y) = 0 \in R_{n,q}$. By the condition on q and p , if c is a valid ciphertext, then $c(u_x, u_y) \bmod q = m(t) + p \cdot e(u_x, u_y) \in R_n$. Thus, we have $m(t) = (c(u_x, u_y) \bmod q) \bmod p$.

See our implementation in Sect. A.

3.1 Security Assumption

Let $\mathfrak{X}(\Gamma_X, R_{n,q}, p)$ be the set of $X(x, y)$ which has a small solution u , that is,

$$\mathfrak{X}(\Gamma_X, R_{n,q}, p) := \{X \in \mathfrak{F}(\Gamma_X, R_{n,q}) \mid \exists u_x, u_y \in R_{n,q,p} : X(u_x, u_y) = 0\}.$$

Akiyama et al. defined the following decision problem:

Definition 3.1 (IE-LWE Problem). For parameters $n, p, q, \Gamma_X, \Gamma_r$, and Γ_{Xr} , we define two sets

$$\begin{aligned} U &:= \mathfrak{X}(\Gamma_X, R_{n,q}, p) \times \mathfrak{F}(\Gamma_{Xr}, R_{n,q}) \\ T &:= \{(X, Xr + e) \mid X \in \mathfrak{X}(\Gamma_X, R_{n,q}, p), r \in \mathfrak{F}(\Gamma_r, R_{n,q}), e \in \mathfrak{F}(\Gamma_{Xr}, R_{n,q,p})\}. \end{aligned}$$

The IE-LWE problem is distinguishing the multivariate polynomials chosen from a ‘noisy’ set T of polynomials from a ‘uniform’ set U .

The IE-LWE assumption states that it is infeasible to solve the IE-LWE problem, where X is chosen by the key-generation algorithm Gen .

Definition 3.2 (IE-LWE Assumption). For parameters $n, p, q, \Gamma_X, \Gamma_r$, and Γ_{Xr} , a key-generation algorithm Gen , and an adversary \mathcal{A} , we define \mathcal{A} ’s advantage as

$$\text{Adv}_{\text{Gen}, \mathcal{A}}^{\text{ie-lwe}}(\kappa) := \left| \Pr \left[\begin{array}{l} X \leftarrow \text{Gen}(1^\kappa); \\ r \leftarrow \mathfrak{F}(\Gamma_r, R_{n,q}); \\ e \leftarrow \mathfrak{F}(\Gamma_{Xr}, R_{n,q,p}); \\ Y := Xr + e; \\ \mathcal{A}(X, Y) \rightarrow 1 \end{array} \right] - \Pr \left[\begin{array}{l} X \leftarrow \text{Gen}(1^\kappa); \\ Y \leftarrow \mathfrak{F}(\Gamma_{Xr}, R_{n,q}); \\ \mathcal{A}(X, Y) \rightarrow 1 \end{array} \right] \right|.$$

We say that the IE-LWE assumption on Gen holds if for any PPT adversary \mathcal{A} , its advantage $\text{Adv}_{\text{Gen}, \mathcal{A}}^{\text{ie-lwe}}(\kappa)$ is negligible in κ .

Akiyama et al. showed that the IEC scheme ($\text{Gen}, \text{Enc}, \text{Dec}$) is IND-CPA secure if the IE-LWE assumption on Gen holds [AGO+18, Theorem 1].

4 Review of Linear-Algebraic Attacks

We review the linear-algebraic attacks in [AGO+18]. In the following, we omit the subscript n from Rot_n and vec_n .

4.1 Key-Recovery Attack

We review the example in the case $\deg X = 1$.

We are given $X(x, y) = a_{00} + a_{10}x + a_{01}y$ and want to find a *small* solution $(u_x, u_y) \in R_{n,q}^2$ satisfying

$$a_{10} \cdot u_x + a_{01} \cdot u_y + a_{00} = 0 \text{ (in } R_{n,q}\text{)}.$$

This implies

$$\text{vec}(u_x) \cdot \text{Rot}(a_{10}) + \text{vec}(u_y) \cdot \text{Rot}(a_{01}) \equiv \text{vec}(-a_{00}) \pmod{q},$$

that is,

$$\left(\text{vec}(u_x), \text{vec}(u_y) \right) \cdot \begin{pmatrix} \text{Rot}(a_{10}) \\ \text{Rot}(a_{01}) \end{pmatrix} \equiv \text{vec}(-a_{00}) \pmod{q}.$$

Therefore, we let

$$A_{\text{kr1}} = [\text{Rot}(a_{10})^\top \mid \text{Rot}(a_{01})^\top] \in \mathbb{Z}_q^{n \times 2n}$$

and consider the lattice

$$\begin{aligned} \Lambda^\perp(A_{\text{kr1}}) &= \{v \in \mathbb{Z}^{2n} \mid v \cdot A_{\text{kr1}}^\top \equiv 0 \pmod{q}\} \\ &= \{(v_x, v_y) \in \mathbb{Z}^{2n} \mid v_x \cdot \text{Rot}(a_{10}) + v_y \cdot \text{Rot}(a_{01}) \equiv 0 \pmod{q}\}. \end{aligned}$$

Now, we consider a target vector $t \in \mathbb{Z}^{2n}$, an arbitrary solution of $t \cdot A_{\text{kr1}}^\top \equiv \text{vec}(-a_{00}) \pmod{q}$. Solving the CVP instance $(\Lambda^\perp(A_{\text{kr1}}), t)$, we obtain a vector $w \in \Lambda^\perp(A_{\text{kr1}})$. We let $\bar{u} = (\text{vec}(u_x), \text{vec}(u_y)) := t - w$.

We have $\bar{u} \cdot A_{\text{kr1}}^\top \equiv \text{vec}(-a_{00}) \pmod{q}$ because $\bar{u} = t - w$. In addition, we expect that the norm of \bar{u} is small, since w is the close vector to t and \bar{u} is the difference.

Remark 4.1. In the case of $\deg X = \deg r = 2$, we have $X(x, y) = a_{00} + a_{10}x + a_{01}y + a_{20}x^2 + a_{11}xy + a_{02}y^2$ and consider a matrix

$$A_{\text{kr2}} = [\text{Rot}(a_{10})^\top \mid \text{Rot}(a_{01})^\top \mid \text{Rot}(a_{20})^\top \mid \text{Rot}(a_{11})^\top \mid \text{Rot}(a_{02})^\top] \in \mathbb{Z}_q^{n \times 5n}.$$

4.2 Message-Recovery Attack

We again review the example in the case $\deg X = 1$ and $\deg r = 1$.

Let us consider $f(x, y) = p \cdot e(x, y) + m \in \mathfrak{F}(\Gamma_{Xr}, R_{n,q})$. The ciphertext c of m has the relation

$$\sum_{(i,j) \in \Gamma_{Xr}} c_{ij} x^i y^j = \left(\sum_{(i,j) \in \Gamma_X} a_{ij} x^i y^j \right) \cdot \left(\sum_{(i,j) \in \Gamma_r} r_{ij} x^i y^j \right) + \left(\sum_{(i,j) \in \Gamma_{Xr}} f_{ij} x^i y^j \right). \quad (2)$$

Let us consider the following matrix

$$A_{\text{mr1}} = \begin{matrix} & 1 & x & y & x^2 & xy & y^2 \\ \begin{matrix} 1 \\ x \\ y \end{matrix} & \begin{pmatrix} A_{00} & A_{10} & A_{01} \\ & A_{00} & & A_{10} & A_{01} \\ & & A_{00} & & A_{10} & A_{01} \end{pmatrix} \end{matrix} \in \mathbb{Z}^{3n \times 6n},$$

where $A_{ij} := \text{Rot}(a_{ij}) \in \mathbb{Z}^{n \times n}$. Let

$$\begin{aligned} \bar{r} &:= (\text{vec}(r_{00}), \text{vec}(r_{10}), \text{vec}(r_{01})) \in \mathbb{Z}^{3n}, \\ \bar{f} &:= (\text{vec}(f_{00}), \text{vec}(f_{10}), \text{vec}(f_{10}), \text{vec}(f_{20}), \text{vec}(f_{11}), \text{vec}(f_{02})) \in \mathbb{Z}^{6n}, \\ \bar{c} &:= (\text{vec}(c_{00}), \text{vec}(c_{10}), \text{vec}(c_{10}), \text{vec}(c_{20}), \text{vec}(c_{11}), \text{vec}(c_{02})) \in \mathbb{Z}^{6n}. \end{aligned}$$

According to Eq. 2, we have

$$\bar{c} \equiv \bar{r} \cdot A_{\text{mr1}} + \bar{f} \pmod{q}.$$

Now, we consider a lattice

$$\Lambda_q(A_{\text{mr1}}) = \{z \in \mathbb{Z}^{6n} \mid z \equiv sA_{\text{mr1}} \pmod{q} \text{ for some } s \in \mathbb{Z}^{3n}\}$$

and a target vector $\bar{c} \in \mathbb{Z}^{6n}$. Solving the CVP instance $(\Lambda_q(A_{\text{mr1}}), \bar{c})$, we obtain $w \in \Lambda_q(A_{\text{mr1}})$. We let $\bar{v} := \bar{c} - w$.

Now, we have $\bar{c} \equiv sA + \bar{v} \pmod{q}$ for some $s \in \mathbb{Z}^{3n}$ and expect that \bar{v} is small. If we obtain $\bar{v} = \bar{f}$, we finally obtain m by taking it modulo p .

Remark 4.2. In the case of $\deg X = \deg r = 2$, we will consider a matrix

$$A_{\text{mr2}} = \begin{matrix} & 1 & x & y & \dots & x^4 & x^3y & x^2y^2 & xy^3 & y^4 \\ \begin{matrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{matrix} & \begin{pmatrix} A_{00} & A_{10} & A_{01} & & & & & & & \\ & A_{00} & & & & & & & & \\ & & A_{00} & & & & & & & \\ & & & A_{00} & & & & & & \\ & & & & A_{20} & A_{11} & A_{02} & & & \\ & & & & & A_{20} & A_{11} & A_{02} & & \\ & & & & & & A_{20} & A_{11} & A_{02} & \end{pmatrix} \end{matrix} \in \mathbb{Z}^{6n \times 15n},$$

and solve the CVP instance with $15n$ -dimensional lattice.

Experimental Results: Akiyama et al. estimate IEC’s security by mounting these attacks against the small parameter sets $n = 10, 20, \dots, 60$ for $\deg X = 1$ and $n = 10, 20, 30, 40$ for $\deg X = 2$. Their environment is

- CPU: AMD Opteron(TM) Processor 848
- Memory: 64 GB
- OS: Linux version 2.6.18-406.el5.centos.plus
- Software: Magma Ver2.21-5

They also define q as small as possible.

They mount a key-recovery attack, which succeeds if and only if $(u_x, u_y) \in R_{n,q,p}$ satisfying $X(u_x, u_y) = 0$ is found. In their experiments, the key-recovery attack for $\deg X = 1$ failed for $n \geq 50$ and that for $\deg X = 2$ failed even for $n \geq 10$.

They also mount a message-recovery attack, which, given X and $Xr + e$, succeeds if and only if $e = (e_1, \dots, e_{6n})$ with $e_i \in [0, p-1]$ is found. The message-recovery attack for $\deg X = 1$ failed for $n \geq 50$. Curiously, the attack for $\deg X = 2$ succeed to find short e even for $n = 40$. (They seem stop their experiment due to time constraint. Their experiment took about 230000 s ≈ 2.7 days to process a 600-dimensional lattice.)

5 Review of Gentry's Attack

We review Gentry's attack against NTRU-Composite [Sil01]. Let us consider NTRU's key generation and encryption: Roughly speaking, we choose a secret key $(f, g) \in R_{n,q,p}^2$ and compute a public key as $h = g/f \in R_{n,q}$. The ciphertext of plaintext $m \in R_{n,q,p}$ with randomness $r \in R_{n,q,p}$ is $c = phr + m \in R_{n,q}$.

Lattice Attack: Coppersmith and Shamir [CS97] pointed out that a short vector $(\text{vec}_n(f), \text{vec}_n(g)) \in \mathbb{Z}^{2n}$ is in a lattice spanned by a matrix

$$L_{CS} := \begin{pmatrix} \text{Rot}_n(1) & \text{Rot}_n(h) \\ \text{Rot}_n(0) & \text{Rot}_n(q) \end{pmatrix} \in \mathbb{Z}^{2n \times 2n}.$$

We have $h = g/f \bmod q$ and this implies $fh + kq = g$ for some $k \in R_n$. Therefore, $(\text{vec}_n(f), \text{vec}_n(k)) \cdot L_{CS} = (\text{vec}_n(f), \text{vec}_n(g))$ as we wanted. Hence, we solve the SVP problem on the lattice and expect to find $(\text{vec}_n(f), \text{vec}_n(g)) \in \mathbb{Z}^{2n}$ as the solution.

Gentry's Attack: Gentry pointed out that there is a ring homomorphism $\theta: R_n \rightarrow R_d$, where $d \mid n$ is a non-trivial divisor.

Theorem 5.1 ([Gen01, Theorem 1]). *Let n be a composite, and d be a non-trivial divisor of n . The mapping*

$$\theta: R_n \rightarrow R_d: f = \sum_{i=0}^{n-1} f_i t^i \mapsto \sum_{i=0}^{d-1} \left(\sum_{j=0}^{n/d-1} f_{jd+i} \right) t^i$$

is a ring-homomorphism.

Gentry considered the $2d$ -dimensional lattice analogue of $\Lambda(L_{CS})$, the lattice spanned by a matrix

$$L_d = \begin{pmatrix} \text{Rot}_d(1) & \text{Rot}_d(\theta(h)) \\ \text{Rot}_d(0) & \text{Rot}_d(q) \end{pmatrix} \in \mathbb{Z}^{2d \times 2d}.$$

The lattice $\Lambda(L_d)$ contains a short vector $(\text{vec}_d(\theta(f)), \text{vec}_d(\theta(g)))$, whose norm is approximately equals to that of $(\text{vec}_n(f), \text{vec}_n(g))$ (see [Gen01, Appendix A.2]). Therefore, we expect the basis-reduction algorithm, say, LLL or BKZ, finds $\theta(f)$ and $\theta(g)$. We can exploit this partial information $\theta(f)$ as follows:

1. Message-Recovery Attack: We have $\theta(f) \cdot \theta(c) = \theta(f) \cdot \theta(m) + p\theta(r) \cdot \theta(g) \pmod q$. Thus, the expected magnitudes of coefficients of $\theta(f) \cdot \theta(m) + p\theta(r) \cdot \theta(g)$ are small, then we can recover $\theta(m)$.
2. Secret-Key Recover Attack: Using $\theta(f)$ and $\theta(g)$ as hint, we again solve the SVP problem and find (f, g) . Indeed, Gentry succeeds to find f in the case of $(n, q, p) = (256, 127, 2)$ in his experiment.

6 Attacks Against Composite n

We employ Gentry's idea. Let us expand the range of the homomorphism $\theta: R_n \rightarrow R_d$ to

$$\theta: R_{n,q}[x, y] \rightarrow R_{d,q}[x, y].$$

6.1 Key-Recovery Attack for $\deg X = 1$

We are given $X(x, y) = a_{01}x + a_{01}y + a_{00}$ and want to find a *small* solution $(u_x, u_y) \in R_{n,q}^2$ satisfying

$$a_{10} \cdot u_x + a_{01} \cdot u_y + a_{00} = 0 \pmod{R_{n,q}}.$$

Applying the homomorphism θ , we have

$$\theta(a_{10}) \cdot \theta(u_x) + \theta(a_{01}) \cdot \theta(u_y) + \theta(a_{00}) = 0 \pmod{R_{d,q}}.$$

Thus, we can try to find $(\theta(u_x), \theta(u_y))$ by using the lattice-basis reduction algorithms on the lattice of dimension $2d$ ($< 2n$).

The concrete attack consists of two sub-attacks, finding $\theta(u_x)$ and $\theta(u_y)$ and finding u_x and u_y by using those hints. The details follow.

Finding $\theta(u_x)$ and $\theta(u_y)$: We set

$$A_{\text{kr1},d} = [\text{Rot}_d(\theta(a_{10}))^\top | \text{Rot}_d(\theta(a_{01}))^\top] \in \mathbb{Z}_q^{d \times 2d}$$

and want to find a short vector v_d satisfying

$$v_d \cdot A_{\text{kr1},d}^\top \equiv \text{vec}_d(-\theta(a_{00})) \pmod q. \quad (3)$$

We consider a lattice $\Lambda_q^\perp(A_{\text{kr1},d})$. Let $t \in \mathbb{Z}^{2d}$ be an arbitrary solution of Eq. 3. We solve the CVP instance $(\Lambda_q^\perp(A_{\text{kr1},d}), t)$ and obtain $w \in \Lambda_q^\perp(A_{\text{kr1},d})$.

Now, we have "short" $\bar{v}_d := t - w$ satisfying Eq. 3. Let us interpret the vector \bar{v}_d as the pair of polynomials $(v_x^{(d)}, v_y^{(d)}) \in R_{d,q}^2$, and assume that $v_x^{(d)} = \theta(u_x)$ and $v_y^{(d)} = \theta(u_y)$.

We have $\text{vol}(\Lambda_q^\perp(A_{\text{kr1},d})) = q^d$, $\gamma \approx \sqrt{2d/(2\pi e)} \cdot \text{vol}(\Lambda_q^\perp(A_{\text{kr1},d}))^{1/2d} = \sqrt{d/\pi e} \cdot q^{1/2}$, and $\|v_d\| \leq 2p\sqrt{2d}$. Since $\gamma \gg \|v_d\|$, that is, the target vector is very shorter than the expected length of the shortest vector, we expect that the LLL/BKZ algorithm can find v_d .

Finding u_x and u_y : We already have a hint $(\theta(u_x), \theta(u_y))$. In this paper, we consider a simpler method than Gentry's one: We set

$$A_{\text{kr1},\text{hint}} = \begin{bmatrix} \text{Rot}_n(a_{10})^\top & \text{Rot}_n(a_{01})^\top \\ \underbrace{I_d \cdots I_d}_{n/d} & \underbrace{I_d \cdots I_d}_{n/d} \end{bmatrix} \in \mathbb{Z}_q^{(n+2d) \times 2n}$$

and try to find a short vector v satisfying

$$v \cdot A_{\text{kr1},\text{hint}}^\top \equiv (\text{vec}_n(-a_{00}), \text{vec}_d(\theta(u_x)), \text{vec}_d(\theta(u_y))) \pmod{q}. \quad (4)$$

We again consider a lattice $\Lambda_q^\perp(A_{\text{kr1},\text{hint}})$. Let $t \in \mathbb{Z}^{2n}$ be an arbitrary solution of Eq. 4. We solve the CVP instance $(\Lambda_q^\perp(A_{\text{kr1},\text{hint}}), t)$ and obtain w . Now, we have a short vector $\bar{v} := t - w$ satisfying Eq. 4.

Interpreting the vector \bar{v} as the pair of polynomials $(u_x, u_y) \in R_{n,q}^2$, we have $a_{10} \cdot u_x + a_{01} \cdot u_y + a_{00} = 0$ in $R_{n,q}$ as we wanted.

We have $\text{vol}(\Lambda_q^\perp(A_{\text{kr1},\text{hint}})) = q^{n+d}$, $\gamma \approx \sqrt{2n/(2\pi e)} \cdot \text{vol}(\Lambda_q^\perp(A_d))^{1/2n} = \sqrt{d/\pi e} \cdot q^{1+d/n}$, and $\|\bar{v}\| \leq p\sqrt{2n}$. Since $\gamma \gg \|\bar{v}\|$, we expect that the LLL/BKZ algorithm can find the target vector \bar{v} .

6.2 Partial-Message-Recovery Attack for $\text{deg } X = 1$

We try to find $\theta(m) \pmod{p}$ from a ciphertext c of m . If so, it easily breaks the IND-CPA security of the IEC scheme.

For simplicity, we define $f(x, y) = pe(x, y) + m$, which results in $\theta(f) = p\theta(e) + \theta(m)$. Since θ is a ring homomorphism from $R_n[x, y] \rightarrow R_d[x, y]$, we have

$$\theta(c) = \theta(r) \cdot \theta(X) + \theta(f).$$

Let us consider the following matrix:

$$A_{\text{pmr1},d} := \begin{matrix} & & & & 1 & x & y & x^2 & xy & y^2 \\ & & & & 1 & A'_{00} & A'_{10} & A'_{01} & & & \\ & & & & x & \begin{pmatrix} A'_{00} & A'_{10} & A'_{01} \\ & A'_{00} & \end{pmatrix} & A'_{10} & A'_{01} & & & \\ & & & & y & & A'_{00} & A'_{10} & A'_{01} & \end{matrix} \in \mathbb{Z}^{3d \times 6d},$$

where $A'_{ij} := \text{Rot}_d(\theta(a_{ij})) \in \mathbb{Z}^{d \times d}$. Let

$$\begin{aligned} \bar{r}_d &:= (\text{vec}_d(\theta(r_{00})), \text{vec}_d(\theta(r_{10})), \text{vec}_d(\theta(r_{01}))) \in \mathbb{Z}^{3d}, \\ \bar{c}_d &:= (\text{vec}_d(\theta(c_{00})), \text{vec}_d(\theta(c_{10})), \text{vec}_d(\theta(c_{01})), \text{vec}_d(\theta(c_{20})), \text{vec}_d(\theta(c_{11})), \text{vec}_d(\theta(c_{02}))) \in \mathbb{Z}^{6d}, \\ \bar{f}_d &:= (\text{vec}_d(\theta(f_{00})), \text{vec}_d(\theta(f_{10})), \text{vec}_d(\theta(f_{01})), \text{vec}_d(\theta(f_{20})), \text{vec}_d(\theta(f_{11})), \text{vec}_d(\theta(f_{02}))) \in \mathbb{Z}^{6d}. \end{aligned}$$

We have

$$\bar{c}_d \equiv \bar{r}_d \cdot A_{\text{pmr1},d} + \bar{f}_d \pmod{q}.$$

Now, we consider a lattice $\Lambda_q(A_{\text{pmr1},d})$ and solve the CVP instance $(\Lambda_q(A_{\text{pmr1},d}), \bar{c}_d)$ and obtain \bar{v}_d . Let us interpret the vector \bar{v}_d as a tuple of polynomials $(v_{00}, v_{10}, v_{01}, v_{20}, v_{11}, v_{02}) \in R_{d,q}^6$. Suppose that we have $\bar{v}_d = \bar{f}_d$, if so, we have $v_{00} = \theta(f_{00})$ and, thus,

$$v_{00} \equiv \theta(f_{00}) \equiv p\theta(e_{00}) + \theta(m) \equiv \theta(m) \pmod{p}.$$

We have $\text{vol}(\Lambda_q^\perp(A_{\text{pmr1},d})) = q^{3d}$, $\gamma \approx \sqrt{6d/(2\pi e)} \cdot \text{vol}(\Lambda_q^\perp(A_d))^{1/6d} = \sqrt{3d/\pi e} \cdot q^{1/2}$, and $\|\bar{v}_d\| \leq (n/d)p^2\sqrt{6d}$. We expect that the LLL/BKZ algorithm can find \bar{v}_d , because $\gamma \gg \|\bar{v}_d\|$.

6.3 Partial-Message-Recovery Attack for $\text{deg } X = 2$

In the case of $\text{deg } X = \text{deg } r = 2$, we consider a matrix

$$A_{\text{pmr2},d} := \begin{matrix} & 1 & x & y & \dots & x^4 & x^3y & x^2y^2 & xy^3 & y^4 \\ \begin{matrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{matrix} & \left(\begin{array}{cccccccc} A'_{00} & A'_{10} & A'_{01} & & & & & & & \\ & A'_{00} & & & & & & & & \\ & & A'_{00} & & & & & & & \\ & & & A'_{00} & & & & & & \\ & & & & A'_{20} & A'_{11} & A'_{02} & & & \\ & & & & & A'_{20} & A'_{11} & A'_{02} & & \\ & & & & & & A'_{20} & A'_{11} & A'_{02} & \end{array} \right) & \in \mathbb{Z}^{6d \times 15d}, \end{matrix}$$

where $A'_{ij} := \text{Rot}_d(\theta(a_{ij})) \in \mathbb{Z}^{d \times d}$. By the similar way, we solve the CVP instance $(\Lambda_q(A_{\text{pmr2},d}), \bar{c}_d)$ and obtain \bar{v}_d , which corresponding to a tuple of polynomials $(v_{00}, v_{10}, \dots, v_{04}) \in R_{d,q}^{15}$. We output $v_{00} \pmod{p}$ as $\theta(m) \pmod{p}$.

We have $\text{vol}(\Lambda_q^\perp(A_{\text{pmr2},d})) = q^{9d}$, $\gamma \approx \sqrt{15d/(2\pi e)} \cdot \text{vol}(\Lambda_q^\perp(A_d))^{1/15d} = \sqrt{15d/2\pi e} \cdot q^{3/5}$, and $\|\bar{v}_d\| \leq (n/d)p^2\sqrt{15d}$. We expect that the LLL/BKZ algorithm can find \bar{v}_d because $\gamma \gg \|\bar{v}_d\|$.

7 Attacks Against Prime n

After reporting the previous attacks to the authors of [AGO+18], they set n as a prime, say, $n = 83$ (and $q = 68339982247$) [Aki17]. In this section, we propose a sub-ring attack, which is applicable to the case that n is a prime.

(Non-trivial) subring: Notice that $R_{n,q}[x]$ is a subring of $R_{n,q}[x, y]$. We consider a ring homomorphism

$$\pi: R_{n,q}[x, y] \rightarrow R_{n,q}[x]: f(x, y) \mapsto f(x, 0).$$

We have the relation $c(x, y) = r(x, y) \cdot X(x, y) + f(x, y)$, where $f(x, y) = pe(x, y) + m$. Applying the ring homomorphism π , we obtain

$$\pi(c) \equiv \pi(r) \cdot \pi(X) + \pi(f) \equiv \pi(r) \cdot \pi(X) + p \cdot \pi(e) + m \pmod{q} \quad (5)$$

and notice that the max norm of $\pi(f)$ is at most that of $f = p \cdot e + m$.

7.1 Message-Recovery Attack Against $\deg X = 1$

Let us recall the message-recovery attack against $\deg X = 1$ in Subject. 4.2. We consider

$$A_{\text{mr1}} := \begin{matrix} & 1 & x & y & x^2 & xy & y^2 \\ \begin{matrix} 1 \\ x \\ y \end{matrix} & \begin{pmatrix} A_{00} & A_{10} & A_{01} \\ & A_{00} & & A_{10} & A_{01} \\ & & A_{00} & & A_{10} & A_{01} \end{pmatrix} \end{matrix} \in \mathbb{Z}^{3n \times 6n},$$

$$\bar{c} := (\text{vec}_n(c_{00}), \text{vec}_n(c_{10}), \text{vec}_n(c_{01}), \text{vec}_n(c_{20}), \text{vec}_n(c_{11}), \text{vec}_n(c_{02})) \in \mathbb{Z}^{6n},$$

where $A_{ij} := \text{Rot}_n(a_{ij}) \in \mathbb{Z}^{n \times n}$, and try to solve the CVP instance $(\Lambda_q(A_{\text{mr1}}), \bar{c})$ to find \bar{f} .

In the lattice-based attacks, we often shorten the basis of the lattice and the target vector to reduce the dimension. Here, we give another approach to shorten them.

Concrete Attack: Deleting the rows and columns whose indices contain y from A and \bar{c} , we obtain

$$A'_{\text{mr1}} := \begin{matrix} & 1 & x & x^2 \\ \begin{matrix} 1 \\ x \end{matrix} & \begin{pmatrix} A_{00} & A_{10} \\ & A_{00} & A_{10} \end{pmatrix} \end{matrix} \in \mathbb{Z}^{2n \times 3n},$$

$$\bar{c}' := (\text{vec}_n(c_{00}), \text{vec}_n(c_{10}), \text{vec}_n(c_{20})) \in \mathbb{Z}^{5n}.$$

Letting

$$\bar{r}' = (\text{vec}_n(r_{00}), \text{vec}_n(r_{10})) \in \mathbb{Z}^{2n},$$

$$\bar{f}' = (\text{vec}_n(f_{00}), \text{vec}_n(f_{10}), \text{vec}_n(f_{20})) \in \mathbb{Z}^{3n},$$

we have

$$\bar{c}' \equiv \bar{r}' \cdot A'_{\text{mr1}} + \bar{f}' \pmod{q},$$

which corresponds to Eq. 5. Thus, solving the CVP instance $(\Lambda_q(A'_{\text{mr1}}), \bar{c}')$, we expect to find \bar{f}' and obtain $m := \text{vec}_n(f_{00}) \pmod{p}$.

Gaussian Heuristic: This shortening reduces the dimension of the lattice from $5n = 415$ to $3n = 249$. We have $\text{vol}(\Lambda_q(A'_{\text{mr2}})) = q^n$ and $\gamma \approx \sqrt{3n/(2\pi e)} \cdot \text{vol}(\Lambda_q(A'))^{1/3n} = \sqrt{3n/2\pi e} \cdot q^{1/3}$ and $\|\bar{f}'\| \leq p^2 \sqrt{3n}$. In our parameter setting, we have $\gamma \approx 380.81$ and $\|\bar{f}'\| \leq 142.02$ and the gap between γ and $\|\bar{f}'\|$ is not so large. Thus it seems hard to find \bar{f}' in this setting.

7.2 Message-Recovery Attack Against $\deg X = 2$

Let us recall the message-recovery attack against $\deg X = 2$ in Subject. 4.2. We consider $A_{\text{mr2}} \in \mathbb{Z}^{6n \times 15n}$ and $\bar{c} := (\text{vec}_n(c_{00}), \text{vec}_n(c_{10}), \text{vec}_n(c_{01}), \dots, \text{vec}_n(c_{04})) \in \mathbb{Z}^{15n}$, and try to solve the CVP instance $(\Lambda_q(A_{\text{mr2}}), \bar{c})$ to find \bar{f} .

Concrete Attack: Deleting the rows and columns whose indices contain y from A and \bar{c} , we obtain

$$A'_{mr2} := \begin{matrix} & 1 & x & x^2 & x^3 & x^4 \\ \begin{matrix} 1 \\ x \\ x^2 \end{matrix} & \begin{pmatrix} A_{00} & A_{10} & A_{20} & & \\ & A_{00} & A_{10} & A_{20} & \\ & & A_{00} & A_{10} & A_{20} \end{pmatrix} \end{matrix} \in \mathbb{Z}^{3n \times 5n},$$

$$\bar{c}' := (\text{vec}_n(c_{00}), \text{vec}_n(c_{10}), \text{vec}_n(c_{20}), \text{vec}_n(c_{30}), \text{vec}_n(c_{40})) \in \mathbb{Z}^{5n}.$$

Letting

$$\bar{r}' = (\text{vec}_n(r_{00}), \text{vec}_n(r_{10}), \text{vec}_n(r_{20})) \in \mathbb{Z}^{3n},$$

$$\bar{f}' = (\text{vec}_n(f_{00}), \text{vec}_n(f_{10}), \text{vec}_n(f_{20}), \text{vec}_n(f_{30}), \text{vec}_n(f_{40})) \in \mathbb{Z}^{5n},$$

we have

$$\bar{c}' \equiv \bar{r}' \cdot A'_{mr2} + \bar{f}' \pmod{q},$$

which corresponds to Eq. 5. Thus, solving the CVP instance $(\Lambda_q(A'_{mr2}), \bar{c}')$, we expect to find \bar{f}' and obtain $m := \text{vec}_n(f_{00}) \bmod p$.

Gaussian Heuristic: We note that this shortening reduces the dimension of the lattice from $15n = 1243$ to $5n = 415$. We have $\text{vol}(\Lambda_q(A'_{mr2})) = q^{2n}$ and $\gamma \approx \sqrt{5n/(2\pi e)} \cdot \text{vol}(\Lambda_q(A'))^{1/5n} = \sqrt{5n/2\pi e} \cdot q^{2/5}$ and $\|\bar{f}'\| \leq p^2 \sqrt{5n}$. In our parameter setting, $\gamma \approx 106330.25$ and $\|\bar{f}'\| \leq 183.35$. We expect that the LLL/BKZ algorithm can find a short vector \bar{f}' because of this large gap.

7.3 Distinguishing Attack for $\deg X = 1$ and $\deg X = 2$

Further, we try to falsify the IE-LWE assumption, that is to distinguish $(X, c) = (X, Xr + e)$ from (X, u) . In order to do so, we try to find a short vector \bar{v}' from $\Lambda_q(A'_{mr1})$. If c is $Xr + e$, then we have $\langle \bar{c}', \bar{v}' \rangle \bmod q$ is “short,” while if c is chosen uniformly at random, then $\langle \bar{c}', \bar{v}' \rangle \bmod q$ is distributed according to the uniform distribution over \mathbb{Z}_q .

This can also be applied to the case of $\deg X = 2$.

8 Experiments

We run our experiment on a virtual machine on our company’s internal private cloud. Our environment is

- CPU: QEMU Virtual CPU version 2.5+
- Memory: 32 GB
- OS: CentOS7 (Linux version 3.10.0-693.5.2.el7.x86_64)
- Software: SageMath version 8.0

8.1 Key-Recovery Attack for $\deg X = 1$

We mount our attack in Subsect. 6.1 with $n = 80$ and $d = 40$. We employ the default BKZ algorithm in SageMath 8.0 as the lattice-basis reduction algorithm and the rounding algorithm to solve the CVP instance. We generate 100 key pairs and try to find a pair $(u_x, u_y) \in R_{n,q,p}^2$ satisfying $X(u_x, u_y) = 0$. In our experiment, 84 secret keys are found from 100 public keys. The attack used an average CPU time of 32.68 s per key on a single core of our server. (min: 29.16, avg: 32.68, med: 32.54, max: 39.11)

We did not check the other settings, say, $d = 20$ or $d = 16$.

8.2 Partial-Message-Recovery Attack for $\deg X = 1$

We mount our attack in Subsect. 6.2 with $n = 80$ and $d = 10$. We employ the default BKZ algorithm with block size 10 as the lattice-basis reduction algorithm and the embedding algorithm to solve the CVP instance. We generate 100 pairs of a public key and a random ciphertext on a random plaintext. In our experiment, all partial message $\theta(m) \bmod p$ are recovered. The attack used an average CPU time of 0.47 s per key on a single core of our server. (min: 0.29, avg: 0.47, med: 0.46, max: 0.73)

8.3 Partial-Message-Recovery Attack for $\deg X = 2$

We mount our attack in Subsect. 6.3 with $n = 80$ and $d = 10$. We employ the default BKZ algorithm as the lattice-basis reduction algorithm and the embedding algorithm to solve the CVP instance. We generate 100 pairs of a public key and a random ciphertext on a random plaintext. In our experiment, all partial message $\theta(m) \bmod p$ are recovered. The attack used an average CPU time of 33.40 s per key on a single core of our server. (min: 20.95, avg: 33.40, med: 32.41, max: 84.77)

8.4 Message-Recovery Subring Attack for $\deg X = 2$

We mount our attack in Subsect. 7.2 with $n = 83$ (and $q = 68339982247$). We employ the BKZ algorithm with options `block_size=10`, `fp="rr"`, `precision=150` as the lattice-basis reduction algorithm and the embedding algorithm to solve the CVP instance. We generate 10 pairs of a public key and a random ciphertext on a random plaintext. In our experiment, all message m are recovered. The attack used an average CPU time of 54842.55 s per key on a single core of our server. (min: 51481.51, avg: 54842.55, med: 54127.69, max: 61770.88)

8.5 Distinguishing Subring Attack for $\deg X = 2$

We mount our attack in Subsect. 7.3 with various prime n with $p = 3$ and a smallest prime q satisfying Eq. 1. We generate 10 public keys on each $n \in$

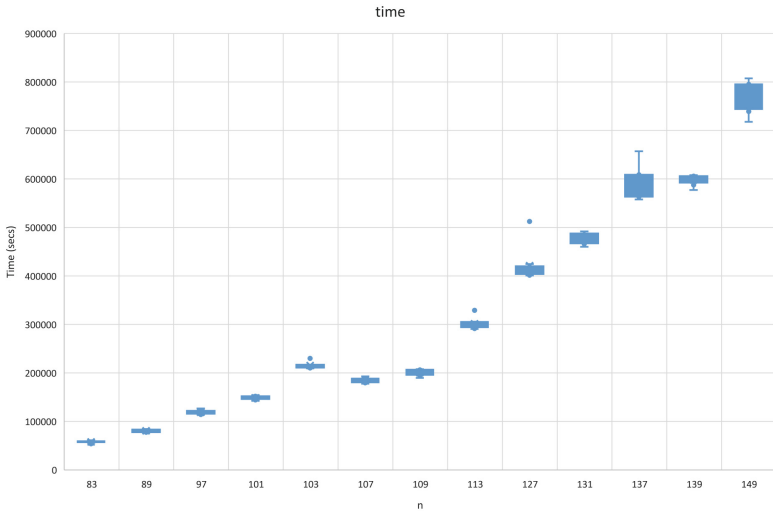


Fig. 1. Summary of running time

{83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149} and try to find a short vector \bar{v}' in the lattice $\Lambda_q(A'_{mr2})$. We employ the BKZ algorithm with options **block_size=10**, **fp="rr"**, **precision=150** up to $n = 113$ and **block_size=10**, **fp="rr"**, **precision=200** for $n \geq 127$ as the lattice-basis reduction algorithm.

The timing results are summarized in Fig. 1 and the qualities of \bar{v}' are summarized in Fig. 2. The attack on $n = 83, 113, 149$ used an average CPU time of 57471.10, 309815.82, 762618.22s per key. The attack on $n = 83, 113$ found short

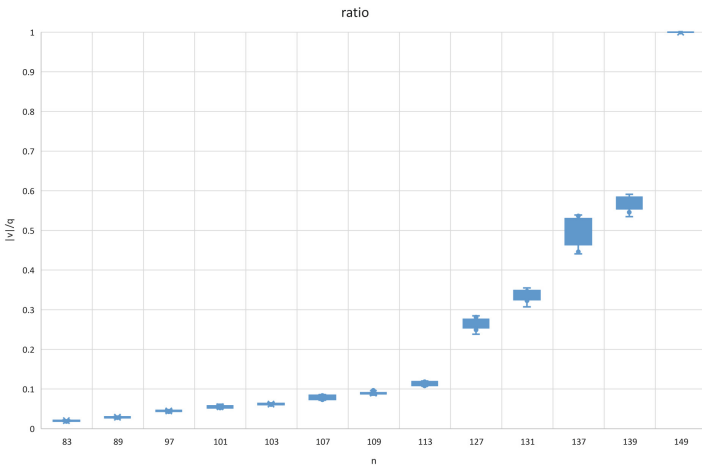


Fig. 2. Summary of ratio $\|\bar{v}'\|/q$

vectors \bar{v}' such that the average of ratio $\|\bar{v}'\|/q$ is 0.021, and 0.11. In the case of $n = 149$, we fail to find short vectors \bar{v}' .

We check the quality of \bar{v}' as follows. We generate 50000 random errors $e_i(x, y) \in \mathfrak{F}(\Gamma_{Xr}, R_{n,q,p})$ and 50000 random polynomials $u_i(x, y) \in \mathfrak{F}(\Gamma_{Xr}, R_{n,q})$. We then compute $\delta_i := \bar{v}' \cdot \bar{e}_i \bmod_c q$ and $\xi_i := \bar{v}' \cdot \bar{u}_i \bmod_c q$, where we denote by \bmod_c the centered modulo operator. We check how they vary.

For example, in the case of $n = 113$, we take the worst vector \bar{v}' with $\|\bar{v}'\|/q = 0.12$. Although this is the worst vector, it is enough to distinguish the errors from uniform as the histogram in Fig. 3 shows.

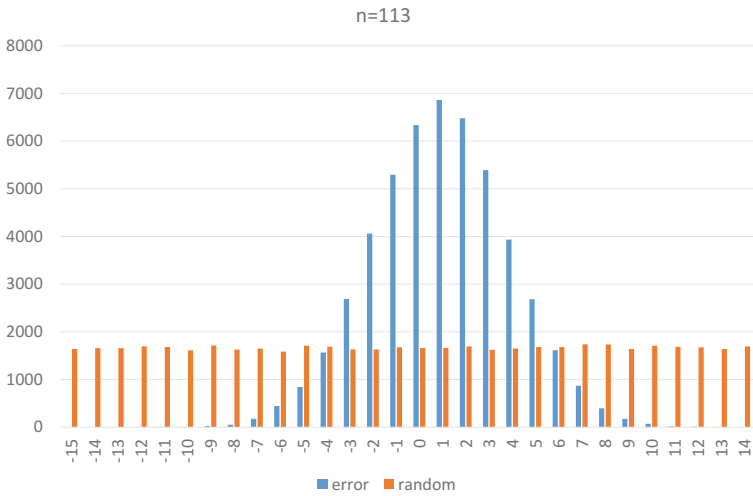


Fig. 3. Histogram of δ_i (blue lines) and ξ_i (orange lines). We count $q/30$ (Color figure online)

9 Conclusion

In this paper, we propose two strategies to reduce the dimension of lattices in lattice-based attacks. The first one is for composite n and is inspired by Gentry’s attack [Gen01] against NTRU Composite [Sil01]. This strategy exploits the ring homomorphism $\theta: R_{n,q}[x, y] \rightarrow R_{d,q}[x, y]$ to reduce the dimension of lattices in lattice-based attacks. The second one is for prime n and exploits another class of subring $R_{n,q}[x]$ of $R_{n,q}[x, y]$ to reduce the dimension. The message-recovery attack succeeds in the case $\deg X = 2$ but fails in the case $\deg X = 1$. The distinguishing attack also succeeds in larger n , say, $n = 113$.

We finally note that we have already reported our attacks to Akiyama et al. and the parameter settings in their paper on Cryptology ePrint Archive [AGO+17b] and NIST PQC submission [AGO+17a] reflected our attacks. They further investigated lattice-based attacks and estimated

the security by following the security-estimation methods of the LWE problems [AGVW17,ADPS16,BDGL15,Che13]. See their paper for details.

Acknowledgment. The author would like to thank Akiyama, Goto, Okumura, Takagi, Nuida, and Hanaoka for their kindness and fruitful discussions. The author would like to thank anonymous reviewers of PQCrypto 2018 for their valuable comments. The author finally would like to thank the XFARM Team for providing their could.

A Implementation

Listing 1.1. ref.sage

```
# Parameters =====
def gen_G(upper_bound, lower_bound):
    # compare with total deg. if equal, (1,0) < (0,1)
    def my_key(a):
        return (a[0] + a[1], a[1], a[0])
    # i for index of x, j for index of y
    l = [(i,j) for j in range(upper_bound+1) \
         for i in range(upper_bound+1) \
         if (lower_bound <= i+j) and (i+j <= upper_bound)]
    return sorted(l, key=my_key)

GX = gen_G(wx,0); Gr = gen_G(wr,0)
GXr = gen_G(wx+wr,0); GXp = gen_G(wx,1)

def bd(n,p):
    return len(GXr) * p * (p-1) * (n * (p-1))^(wx+wr)

q = next_prime(bd(n,p))

# Rings =====
Zq = Integers(q)
R.<t> = Zq[]
Rq = R.quotient(t^n-1)
Rqd = R.quotient(t^d-1)
F.<x,y> = Rq[]

# Random polys =====
def random_tpoly(p): return R([randint(0,p-1) for _ in range(n)])

def random_template(p,indices):
    a = 0
    for (i,j) in indices:
        a += Rq(random_tpoly(p)) * x^i * y^j
    return a
```

```

def random_r(): return random_template(q,Gr)
def random_e(): return random_template(p,GXr)

# Cryptosystem =====
def skgen():
    return random_tpoly(p), random_tpoly(p)

def pkgen(ux,uy):
    X = random_template(q,GXp)
    X -= X(ux,uy)
    return X

def encrypt(X,m):
    return Rq(m)+ X * random_r() +p * random_e()

def decrypt(ux,uy,c):
    cu = c(ux,uy)
    mt = cu.lift().change_ring(ZZ).change_ring(Integers(p))
    # output mt in Rq
    return mt.change_ring(Integers(q))

```

References

- [ADPS16] Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: Holz, T., Savage, S. (eds.) USENIX Security Symposium 2016, pp. 327–343. USENIX Association (2016). <https://eprint.iacr.org/2015/1092>
- [AG06] Akiyama, K., Goto, Y.: A public-key cryptosystem using algebraic surfaces. In: PQCrypto 2006, pp. 119–138 (2006). <http://postquantum.cr.jp.to/>
- [AGM09] Akiyama, K., Goto, Y., Miyake, H.: An algebraic surface cryptosystem. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 425–442. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_24
- [AGO+17a] Akiyama, K., Goto, Y., Okumura, S., Takagi, T., Nuida, K., Hanaoka, G., Shimizu, H., Ikematsu, Y.: Giophantus. Technical report, National Institute of Standards and Technology (2017). <https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions>
- [AGO+17b] Akiyama, K., Goto, Y., Okumura, S., Takagi, T., Nuida, K., Hanaoka, G., Shimizu, H., Ikematsu, Y.: A public-key encryption scheme based on non-linear indeterminate equations (Giophantus). Cryptology ePrint Archive, Report 2017/1241 (2017). <https://eprint.iacr.org/2017/1241>
- [AGO+18] Akiyama, K., Goto, Y., Okumura, S., Takagi, T., Nuida, K., Hanaoka, G.: A public-key encryption scheme based on non-linear indeterminate equations. In: Adams, C., Camenisch, J. (eds.) SAC 2017. LNCS, vol. 10719, pp. 215–234. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72565-9_11

- [AGVW17] Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving uSVP and applications to LWE. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 297–322. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_11
- [Aki17] Akiyama, K.: Private communication, 04 October 2017
- [BDGL15] Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. IACR Cryptology ePrint Archive 2015:1128 (2015)
- [Che13] Chen, Y.: Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe. Ph.D. thesis, Thèse de doctorat dirigée par Nguyen, Phong-Quang Informatique Paris 7 (2013)
- [CS97] Coppersmith, D., Shamir, A.: Lattice attacks on NTRU. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 52–61. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_5
- [Gen01] Gentry, C.: Key recovery and message attacks on NTRU-composite. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 182–194. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_12
- [GPV08] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.) STOC 2008, pp. 197–206. ACM (2008). <https://eprint.iacr.org/2007/432>
- [Sil01] Silverman, J.H.: Wraps, gaps, and lattice constants. Technical report 11, version 2, NTRU Cryptosystems (2001)

Hash-Based Cryptography



Grafting Trees: A Fault Attack Against the SPHINCS Framework

Laurent Castelnovi¹, Ange Martinelli², and Thomas Prest^{2(✉)}

¹ Alten Sud-Ouest, Labège, France

laurent.castelnovi@live.fr

² Thales Communications & Security, Gennevilliers, France

{ange.martinelli,thomas.prest}@thalesgoup.com

Abstract. Because they require no assumption besides the preimage or collision resistance of hash functions, hash-based signatures are a unique and very attractive class of post-quantum primitives. Among them, the schemes of the SPHINCS family are arguably the most practical stateless schemes, and can be implemented on embedded devices such as FPGAs or smart cards. This naturally raises the question of their resistance to implementation attacks.

In this paper, we propose the first fault attack against the framework underlying SPHINCS, GRAVITY-SPHINCS and SPHINCS⁺. Our attack allows to forge any message signature at the cost of a single faulted message. Furthermore, the fault model is very reasonable and the faulted signatures remain valid, which renders our attack both stealthy and practical. As the attack involves a non-negligible computational cost, we propose a fine-grained trade-off allowing to lower this cost by slightly increasing the number of faulted messages. Our attack is generic in the sense that it does not depend on the underlying hash function(s) used.

1 Introduction

Hash-based signatures base their security solely on the hardness of finding collisions or (second) preimages for hash functions, and do not require any additional assumption. This striking property makes them stand out even among other post-quantum schemes. From a strict viewpoint of security assumptions, one can hardly expect better as Rompel [Rom90] has shown that secure signatures exist if and only if one-way functions exist, and Song [Son14] has extended this result to quantum adversaries. In addition, hash-based signatures are easy to analyze and their security does not depend on the choice of the underlying primitive.

Since Lamport [Lam79] proposed the first hash-based signature scheme – which could sign only one message –, several constructions have been proposed to improve its efficiency. They can be separated in two classes: stateful and stateless constructions. Stateful signatures, introduced by Merkle [Mer90], constrain the

Large parts of this work were done when Laurent Castelnovi was an intern at Thales.

signer to maintain a record of its used keys. Such a requirement may generate operational problems, in particular when the key is used by multiples servers. Stateless signatures, as introduced by Goldreich [Gol86], lift this requirement but at the cost of a huge blow-up in the signature time and size.

It is only recently that practical stateless constructions have been proposed. In 2015, Bernstein et al. [BHH+15] proposed SPHINCS, a hash-based signature scheme which achieved both statelessness and a reasonable efficiency (with respect to the running time and signature size) by combining the constructions of Goldreich and Merkle, and using a few-time signature scheme. In 2017, two variations of SPHINCS were proposed to NIST’s call for post-quantum cryptographic schemes [NIS16]: GRAVITY-SPHINCS [AE17b] by Aumasson and Endignoux, and SPHINCS⁺ [BDE+17] by Bernstein et al.

Hash functions can be implemented very efficiently on constrained devices and it is not surprising that several implementations of hash-based signatures on micro-controllers have been proposed [RED+08, HBB12], including an ARM implementation [HRS16] of SPHINCS. However, embedded devices are known to be sensitive to physical attacks such as side-channel analysis or fault attacks.

Since the seminal article of Boneh et al. [BDL97], fault attacks have proved to be the strongest kind of cryptanalysis on embedded devices. In a fault attack, we suppose that the attacker is strong enough to corrupt the internal state of an algorithm during its execution. While this supposes a rather powerful attacker, these conditions can often be fulfilled in real life and generally result in devastating attacks. However, to the best of our knowledge, no fault attack against hash-based signatures has been publicly proposed.

1.1 Our Contribution

At a very high level, the SPHINCS framework (in this document, this notion encompasses the original SPHINCS scheme, as well as GRAVITY-SPHINCS and SPHINCS⁺) combines hash trees and several one-time signature schemes (OTS) inside a tree data structure in order to obtain a stateless signature scheme. We propose the first fault injection attack against the SPHINCS framework. The attack is done in two steps, a faulting part and a grafting part:

1. *The faulting step.* Two signatures for the same message are queried. During the second signature computation, a fault is provoked so that an OTS inside the SPHINCS framework ends up signing a different value than the first time. Usually, an OTS key is only used to sign a single value, but our fault attack compels it to do otherwise.
2. *The grafting step.* We show that the knowledge of the two signatures – the correct one and the faulted one – can be exploited to recover parts of the secret key of the OTS which was subjected to the fault, and therefore to partially compromise it. In turn, an attacker will use this compromised OTS as a mean to authenticate a tree different from the one it is supposed to authenticate.

The attacker then generates a tree which is entirely under its control, and will use the compromised OTS to *graft* it to the SPHINCS tree, which is why we call this step the grafting step.

The grafted tree is chosen by the attacker and independent from the secret key, while allowing to generate valid signatures for some messages. We show that it is more than enough to provide universal forgery ability to an attacker while explaining how she can achieve it. The attack requires little power from the attacker – which makes it practical – and produces valid signatures, which renders it particularly stealthy.

Whereas this attack comes with a non-negligible computational cost for each forgery, we propose trade-offs to lower this cost by slightly increasing the number of faulted signatures available to the attacker. Our attack is generic in the sense that it targets the SPHINCS framework: it is successful regardless of the underlying hash function used, and is indifferent to the specificities of the original SPHINCS, GRAVITY-SPHINCS or SPHINCS⁺.

1.2 Roadmap

First we will introduce the notions related to trees. In Sect. 2, we will give a quick overview of hash-based signatures constructions. Then we will describe our attack in Sect. 3. The grafting step will be presented before the faulting step, as it only requires two signatures by the same OTS and is indifferent to whether they were obtained through a fault attack. We will then discuss countermeasures in Sect. 3.4. Section 4 will conclude this paper and expose open questions.

1.3 Related Works

Our grafting technique relies on and extend a result by Bruinderink and Hülsing [GBH16] about the security of common OTS’s under two-message attacks.

Due to their relative novelty, the resistance of post-quantum cryptographic schemes against fault attacks has only recently been studied. A wide array of attacks against lattice-based schemes has been covered in [BBK16], and a loop-abort attack has been demonstrated in [EFGT16]. For schemes based on supersingular isogenies, loop-abort and point decompression attacks have been investigated in [BG15, Ti17, GW17]. While we know of no fault attack against hash-based signatures, countermeasures have been studied in [MKAA16].

Hash functions have been targeted by fault attacks on their keyed operation modes. Notably Hemme and Hoffmann [HH11] propose a differential fault analysis allowing the attacker to recover the internal state of a SHA-1 instance using about 1000 faulted hashes with the fault targeting a specific variable in the computation. A similar attack targeting SHA-3 was presented in 2015 [BGS15]. This attack involves random single bit faults on 80 messages to recover most of the SHA-3 internal state. In comparison, our attack requires only one fault, and the precision needed by the attacker in order to succeed is very low.

2 Preliminaries

We first set up the notations, then introduce the security models used for signature schemes and present Merkle's and Goldreich's constructions.

2.1 Notations and Conventions

We denote by λ the security parameter of a signature scheme. $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ denotes a cryptographic hash function. We will note vectors in bold lowercase. Whenever we consider faulting the value of a vector \mathbf{v} , we denote by \mathbf{v}^* the faulted value of \mathbf{v} .

2.2 Dendrologic Notations

We recall notions related to trees. We suppose that the definitions of (balanced) binary tree, parent, child, sibling, root, leaf and internal node are known.

We denote by $\&f$ the address of a leaf f . The height of a tree is the length of the longest path between the root and any node. Two nodes are at the same height (resp. in the same layer, resp. at the same level) if they lie at the same distance from the root.

In this article, we also deal with hypertrees, which are trees whose nodes are trees. The height of a hypertree is the sum of the largest heights of each layer. To avoid confusion between the hypertree and the node trees, we will refer to the *layers* of the hypertree and the *levels* of the node trees.

As an example, Fig. 3 depicts a toy version of a SPHINCS hypertree. This example has 2 layers of height 2, hence it has a total height of 4. SPHINCS-256 has 12 layers of height 5, so its total height is 60.

2.3 Security Models for Signature Schemes

We briefly recall some classical security notions for signature schemes.

Definition 21. *Existential forgery* – An adversary is able of existential forgery if there exists a message m such that she can exhibit a valid (message, signature) pair (m, σ^*) where σ^* was not produced by the legitimate signer.

Definition 22. *Universal forgery* – An adversary is able of universal forgery if for any message m , she can exhibit a valid signature σ^* .

Any attacker able of universal forgery is able of existential forgery. For more formal notations, we refer the reader to e.g. [GBH16].

2.4 Hash-Based Signatures

Hash-based signatures stem from a very simple idea: the public key is a commitment of the secret key, whereas the signature of a message consists of revealing some information from which the verifier can recompute the commitment.

A simple way to build a hash-based one-time signature (OTS) can be defined as follows. Given a hash function H , let a secret key $S = (S_1, S_2)$ and the message space be $\llbracket 0, M - 1 \rrbracket$ for an integer M . The public key is

$$P = (P_1, P_2) = (H^M(S_1), H^M(S_2)).$$

The signature of a message $m \in \llbracket 0, M - 1 \rrbracket$ is

$$\sigma = (\sigma_1, \sigma_2) = (H^m(S_1), H^{M-m}(S_2)).$$

The verifier only needs to check that

$$(\sigma_1^{M-m}, \sigma_2^m) = (P_1, P_2).$$

This scheme is one-time if H is preimage-resistant. However, it is not two-time, since given signatures for messages $m_1 < m_2$ one can compute signatures for any $m_1 < m_3 < m_2$, thus breaking existential unforgeability.

We draw the readers' attention to the fact that in the scheme we presented, the public key can be computed from any valid signature. This is a common feature among hash-based signatures and will effectively be the case for all the schemes considered in this paper. From this feature, one does not need the public key if it is able to authenticate it. It allows to derive many-times signatures schemes from OTS by computing signature trees.

In the rest of this section, we present an OTS, two few-times signatures (FTS) and two many-times signatures, a stateful one and a stateless one.

2.4.1 An OTS: WOTS

WOTS is a one-time signature whose principle was enunciated by Merkle [Mer90] following an idea from Winternitz. It is parameterized by three values:

- w : the size of the words used by WOTS
- ℓ_1 : the fixed number of words of size w of the messages to be signed
- ℓ_2 : the fixed number of words of size w of the parity-check value used in the signature algorithm.

Let $\ell = \ell_1 + \ell_2$ be the fixed number of words of size w of the signature. We can now detail WOTS.

- **Keygen()**
 1. Let $\text{sk} = (s_i)_{i=1, \dots, \ell}$ where the s_i are uniformly random w -bits words;
 2. For $1 \leq i \leq \ell$, $p_i \leftarrow H^{w-1}(s_i)$;
 3. *public key*: $\text{pk} \leftarrow (p_1, \dots, p_\ell)$, *private key*: sk .
- **Sign(m, sk)**
 1. Express m in base w : $m = (m_1 m_2 \dots m_{\ell_1})_w$;
 2. Compute the parity-check value $C \leftarrow \sum_{i=1}^{\ell_1} (w - 1 - m_i)$;
 3. Express C in base w : $C = (C_1 C_2 \dots C_{\ell_2})_w$;

4. $\mathbf{b} = (b_1, b_2, \dots, b_\ell) \leftarrow (\mathbf{m}_1, \dots, \mathbf{m}_{\ell_1}, C_1, \dots, C_{\ell_2})$ – we will later call it the *b-vector* of \mathbf{m} ;
 5. For $1 \leq i \leq \ell$, $\sigma_i \leftarrow H^{b_i}(s_i)$;
 6. *signature*: $\sigma \leftarrow (\sigma_1, \dots, \sigma_\ell)$.
- **Verify**($\mathbf{m}, \sigma, \text{pk}$)
 1. Compute the *b-vector* of \mathbf{m} as in the signature algorithm (Steps 1–4);
 2. Accept if and only if $\forall i \in \llbracket 1, \ell \rrbracket, p_i = H^{w-1-b_i}(\sigma_i)$.

Remark 1. GRAVITY-SPHINCS implements the unmasked version of WOTS described above, but SPHINCS⁽⁺⁾ replaces WOTS by a variant, WOTS+, which uses random masks in order to replace collision resistance by second preimage resistance. Since our attack is indifferent to the presence of masks, we present it only in the case of the mask-less scheme (WOTS) as it makes our exposition simpler.

Parameters: In practice, SPHINCS-256 (it is the practical instantiation of SPHINCS proposed in [BHH+15]), GRAVITY-SPHINCS and SPHINCS⁺ set:

$$\begin{cases} \ell_1 = 64, \\ \ell_2 = 3, \\ w = 16. \end{cases}$$

These parameters offer a good trade-off between size and speed and are usually chosen in the most recent constructions.

In [GBH16, Sect. 5], the authors study (among other scenarii) WOTS resistance against existential forgery under two-random-message attacks. They argue that the probability of being able to forge the signature of a random message \mathbf{m}_3 knowing the signature of two known random messages \mathbf{m}_1 and \mathbf{m}_2 is roughly equal to the probability that for all $0 \leq i < \ell$, the i -th coordinate of the *b-vector* of \mathbf{m}_3 is lower than the i -th coordinate of the *b-vector* of \mathbf{m}_1 or \mathbf{m}_2 . We will see that this existential forgery on WOTS can be extended to a universal forgery on the SPHINCS framework in Sect. 3.1.

2.4.2 FTS

In order to expand an OTS construction to a FTS signature scheme, one can generate many OTS, link the public keys using an authentication tree and use the root of this tree as public key. To sign a message, the signer only needs to choose a subset of the OTS generated and sign the message with each of them. The verifier only has to recover the various public keys from the signatures and to check if the public key is equal to the authentication value associated with the public keys and the corresponding authentication path.

All three algorithms based on the SPHINCS framework make use of different FTS. In the context of our attack, one only needs to keep two facts in mind:

- just like WOTS, the FTSs are entirely deterministic;
- in each of these FTSs, the public key can be directly computed from a valid signature.

2.4.3 A Stateful Construction: Merkle’s Scheme

Merkle’s scheme [Mer90] is based on hash trees, which are (generally balanced) binary trees in which each internal node is defined as the hash of its two concatenated child nodes. In Merkle’s construction, each leaf of a hash tree is an OTS public key: such a hash tree is called a *Merkle tree*. The public key for Merkle’s scheme is the root of the Merkle tree and the private key is the set of all the OTS private keys paired with the OTS public keys.

For a leaf f of a Merkle tree, we denote by $A(f)$ and call authentication path of f , the unique set of nodes (with one node per level, excluding the root) such that the root of the Merkle tree can be recomputed from f and $A(f)$.

To sign a message, the signer chooses an unused OTS key pair (sk_i, pk_i) in a leaf of the Merkle tree: he signs m with sk_i and sends the signature together with pk_i and its authentication path $A(pk_i)$. The receiver verifies that: (1) the message’s signature using the OTS is valid, and (2) the general public key (which is the root of the Merkle tree) can be recomputed from pk_i and $A(pk_i)$.

This scheme has two major drawbacks. First, the signature time – or memory requirement – is exponential in the tree height, since the whole tree must either be stored or recomputed each time a signature is performed.¹ Second, the signer must keep track of the used OTS key pairs, which makes the scheme stateful.

2.4.4 A Stateless Construction: Goldreich’s Signature

Goldreich’s proposal [Gol86] solves the two aforementioned issues: it is still based on a binary tree whose leaves are OTS public keys, but internal nodes are now OTS key pairs. Each node of the tree is uniquely indexed by a bitstring which is used, together with a seed which is part of the overall private key, to pseudo-randomly generate the node’s key pair.

The scheme’s public key is the root’s public key and its private key is composed of the root’s private key and the seed referred to above. To sign a message, one randomly selects a leaf and then signs the message with this key pair. Each node (specifically the public key inside) between this leaf and the root is then signed, together with its sibling node, by its father node. The verifier accepts if and only if all the signatures are valid.

The drawback of this approach is the signature size. For 128 bits of pre-quantum security, one needs a 256-layer tree; using for example a Winternitz OTS with parameter $w = 16$ (see Sect. 2.4.1) and a hash function with a 256-bit output, the signature size reaches 1.65 MB.

2.5 SPHINCS

The aim of SPHINCS is twofold: to achieve moderate signature time and size, and to get rid of any kind of state. To reach this goal, the SPHINCS tree is designed as a Goldreich tree whose nodes are Merkle trees.

¹ There exist techniques which get rid of exponential running time at the expense of somewhat increasing state size, such as the tree traversal algorithm of [BDS08].

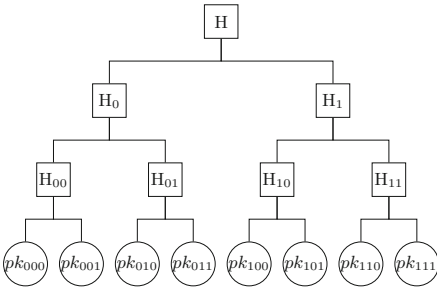


Fig. 1. A Merkle tree. Two merging arrows mean “the parent is the hash of the two children”. The authentication path of $f = pk_{000}$ would be $A(f) = \{pk_{001}, H_{01}, H_1\}$.

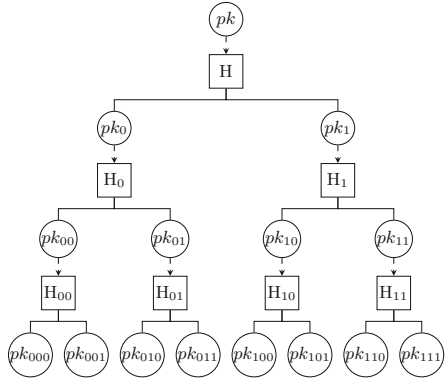


Fig. 2. A Goldreich tree. In addition to the notations of Fig. 1, a dashed arrow means “the leaf signs the root below”.

In this new configuration, each leaf in a Merkle tree is used to sign the root of a Merkle tree located in the layer below. Such a construction can also be found in GMSS [BDK+07] and XMSS [BDH11, HRB13]. Moreover, leaves of a SPHINCS tree sign a public key of a few-time signature (FTS) scheme, which security is not compromised if the same key pair is used on few different messages. This is summarized in Fig. 3.

In order to have a quick overview of SPHINCS, one can see it as a combination of 3 types of trees. Namely:

1. The SPHINCS hypertree: a Goldreich tree of height h (60 in SPHINCS-256) organised in d layers (12 in SPHINCS-256). Each layer’s leaf signs the root of a Merkle tree.
2. Merkle trees of size h/d ($= 5$ in SPHINCS-256) whose leaves are public keys for the OTS used in the Goldreich construction: WOTS.
3. The FTS used to sign the message is signed by the last layer of the hypertree.

We now delve a bit deeper into SPHINCS’s machinery.

A SPHINCS tree of height h can be seen as a Goldreich tree of d layers with Merkle trees of height h/d instead of nodes. In [BHH+15], a few modifications have been made to the Merkle tree construction described in Sect. 2.4.3. One of them is important for our work: in the leaves of SPHINCS’s Merkle subtrees, all WOTS public keys are compressed as follows: their ℓ parts are considered as leaves of a binary hash tree; this tree’s root is then computed applying this rule: if a node has no sibling, then it is lifted to a higher level in the tree until it has one. The tree’s root stands as the compressed WOTS public key.

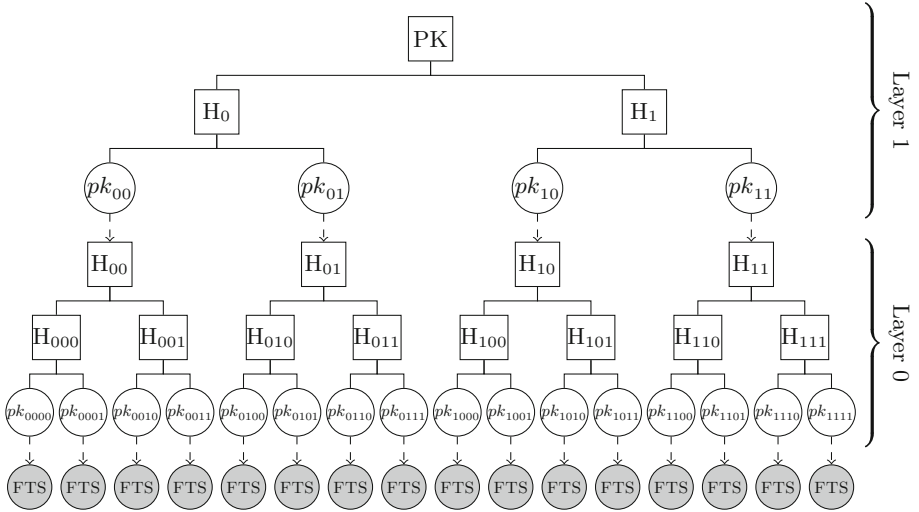


Fig. 3. A SPHINCS hypertree of height 4, which can be seen as a Goldreich tree of two layers with Merkle trees of height 2 in node places. In addition to the notations of Figs. 1 and 2, gray discs denote FTS instances (HORST for the original SPHINCS, PORST for GRAVITY-SPHINCS, FORS for SPHINCS⁺).

Like in Goldreich’s construction, where each node is indexed, each *leaf* has an address in SPHINCS, which contains its layer in the SPHINCS hypertree, the number of its Merkle tree in the layer and its position in the Merkle tree.

We now describe the original SPHINCS signature scheme (which we will call o-SPHINCS to disambiguate it from the SPHINCS framework).

- **Keygen()**

Pick a pair of seeds $(S_1, S_2) \in \{0, 1\}^\lambda \times \{0, 1\}^\lambda$ at random and generate the top Merkle tree (the one in layer $d - 1$), whose root is the overall public key pk . The private key is $sk \leftarrow (S_1, S_2)$.

- **Sign(m, sk)**

1. Generate 2 pseudo-random values $(R_1, R_2) \in (\{0, 1\}^\lambda)^2$ from m and S_2 ;
2. Compute $D = H(R_1 || m)$;
3. $idx \leftarrow$ the h leftmost bits of R_2 ;
4. Generate the HORST key pair of index idx ;
5. $\sigma_H =$ signature of D using this HORST key pair;
6. $\sigma_0 =$ signature of the HORST public key using the WOTS key pair at layer 0, which is (in compressed form) in the leaf f_0 with $\&f_0 = d || idx$;
7. For $1 \leq i < d$, $\sigma_i =$ signature of the root of the Merkle tree containing f_{i-1} , using the WOTS key pair in the appropriate leaf f_i of the layer i ;
8. *signature*: $\sigma = (idx, R_1, \sigma_H, \sigma_0, A(f_0), \sigma_1, A(f_1), \dots, \sigma_{d-1}, A(f_{d-1}))$.

- Verify(m, σ, pk)
 1. Compute $D = H(R_1 || m)$;
 2. Compute the HORST public key assuming σ_H is valid;
 3. For $0 \leq i < d$:
 - (a) assume that the WOTS signature σ_i is valid and deduce a WOTS public key from it and from the root computed the step before²;
 - (b) assume that $A(f_i)$ is correct and compute the root of its Merkle tree;
 4. compare this last root to the SPHINCS public key: accept if and only if they are equal.

2.6 Gravity-SPHINCS and SPHINCS⁺ Modifications

GRAVITY-SPHINCS was proposed in [AE17b], with several changes to O-SPHINCS aiming at improving its performance and signature size. The changes relevant to our attack are the following:

1. The top layer of the hypertree is now cached as it is always used in the signature algorithm, and its height is increased from 5 to 20 (thus lowering the number of layers in the hypertree). As a result, the number of leaves in the topmost Merkle tree is increased from 32 to 2^{20} .
2. The index of the FTS instance is now derived directly from the message and a public salt computed by the signer from its secret key (this is well summarized in [AE17a, Fig. 3]). As a consequence, the verifier can verify the index and the attacker cannot choose it anymore – but we will see that it is easy to get around this protection.

Independently, SPHINCS⁺ was proposed in [BDE+17]. The modification relevant to our attack is that the message digest md and FTS index idx are computed as

$$(md || idx) \leftarrow H(r, pk, m), \quad (1)$$

where r is a public salt generated by the signer from the message and a private seed. This change in index generation is similar to the one of GRAVITY-SPHINCS. For simplicity, this document will only focus on the parameter sets targeting NIST’s security level 1.

We will see that these modifications, while theoretically increasing the cost of our attack, actually have a very limited impact on its efficiency.

Parameters. O-SPHINCS, GRAVITY-SPHINCS and SPHINCS⁺ propose parameters to provide 128 bits of *quantum* security against existential forgery (assuming 256-bits messages). Table 1 summarizes these parameters.³ We note that [AE17b] proposed several trade-offs between efficiency and signature size, as well as variations on the number of signatures allowed by the context.

² Which is the value whose signature is σ_i .

³ We choose the NIST-oriented version of GRAVITY-SPHINCS according to [AE17b].

Table 1. Parameters for O-SPHINCS, GRAVITY-SPHINCS and SPHINCS⁺

Scheme	Security	w	ℓ	h	d	Sig. size (kB)
O-SPHINCS	128	16	67	60	12	41
GRAVITY-SPHINCS	128	16	67	60	6	27
SPHINCS ⁺ -128f	128	16	67	60	20	17
SPHINCS ⁺ -128s	128	16	67	64	8	8

3 A Grafting Attack Against the SPHINCS Framework

In this section we propose a new kind of attack against the hyper-tree structure of the SPHINCS framework. The goal of this attack is to insert a branch under our control below a leaf f_{d-1} (which is an OTS public key) of the top layer. In order to do this, one must be able to provide a signature for the root of the branch which is valid for the key f_{d-1} . Once the root is authenticated – and the branch grafted, one has total control over the branch and can easily modify any of the nodes inside, both by modifying the seed used for its generation and by randomizing unverifiable values.

In the rest of this section we will detail the principles of the grafting attack and its implications in terms of security. Finally, we will provide a practical fault attack that leads to a universal forgery on the SPHINCS framework, and we will discuss different complexity trade-offs. At last we will provide a short overview of the possible countermeasures to our attack.

3.1 Grafting a Branch in the SPHINCS Hyper-Tree

Let us target a leaf f_{d-1} of the top layer of the hyper-tree. We suppose that the corresponding WOTS key has signed two different values, which the attacker knows along with their signatures. According to [GBH16], she is able to forge a WOTS signature with a probability $p_w \approx 2^{-34}$. We convert this existential forgery capability against WOTS into the samea universal forgery capability against any SPHINCS-style scheme. In order to find a forgeableforge a message m , we proceed as follows:

1. Randomly generate a seed such that the index of the FTS to be used falls under the targeted WOTS instance. It happens with probability p_F , where p_F is 1 for O-SPHINCS,⁴ 2^{-20} for GRAVITY-SPHINCS,⁵ and 2^{-3} (resp. 2^{-8}) for SPHINCS⁺-128f (resp. -128s).
2. From this seed and the message m , compute the signature up to the penultimate layer of the hyper-tree. With probability p_w , the root of the layer can be signed by the attacker capacity on the corresponding WOTS signature.

⁴ O-SPHINCS provides the verifier no mechanism to check that the FTS index is valid. An attacker can therefore directly pick a suitable index, hence the probability 1.

⁵ The probability to find such a seed is equal to the inverse of number of leaves in the top-most layer of the hyper-tree, which is 2^{-20} for GRAVITY-SPHINCS.

3. Complete the signature using the legitimate authentication path of the signer, known from the legitimate signature of any message whose authentication path in the hyper-tree goes through f_{d-1} .

The naive way to convert the existential forgery described above into universal forgery achieve the forgery described above is to randomly choose the seed (from which are generated the OTS and all the FTS used during the signature) in order to fulfill two requirements:

1. the FTS used is under the targeted WOTS instance: this happens with the probabilities p_F stated in Sect. 3.1.
2. the attacker can sign the root of the Merkle's tree used in the penultimate layer with the targeted WOTS secret key: this happens with probability p_W .

To find a seed which simultaneously fulfills both requirements, an attacker needs to try about $1/p_F p_W$ seeds for each message. These trials can be done entirely offline. We note that the number of hash computations is even higher as every trial costs around 2^{15} hashes. However, it is possible to do better.

Indeed, even though an honest signer generates (the OTS secret keys corresponding to the leaves of) Merkle trees with a private seed, there is no way a verifier can check that this was effectively the case. Therefore, the search of a suitable Merkle tree for the penultimate layer (by suitable, we mean that its root can be signed with the targeted WOTS key) can be *decorrelated* from the search for a suitable FTS index. This makes the number of trials drop from $1/p_F p_W$ to $1/p_F + 1/p_W$.

In addition, a signature does not contain whole Merkle trees but only, for each of them, a leaf f_i ⁶ and its authentication path $A(f_i)$; this reduces signature size as well as verification time. However, it also allows to speed up forgery as the attacker does not have to generate a suitable Merkle tree but only a leaf f_{d-2} and an authentication path $A(f_{d-2})$ which looks like an authentication path in a suitable Merkle tree. To do this, the attacker can simply choose all the values of $A(f_{d-2})$ at random, except the last one. She then tries several values for this last value, until the root computed from f_{d-2} and $A(f_{d-2})$ can be signed with f_{d-1} . With this improvement, each new trial now costs *one* hash instead of 2^{15} hashes.

With these improvements, the cost of a forgery on any SPHINCS scheme drops from up to $2^{15}/p_F p_W$ hashes down to $1/p_F + 1/p_W$ hashes. As an illustration, this represents a drop from 2^{69} to 2^{34} for GRAVITY-SPHINCS.

3.2 Fault Injection Against the SPHINCS Framework

As we have been seen before, the entire attack depends on the capability of the attacker to obtain two distinct WOTS signatures for the same secret key. In the context of the SPHINCS framework, the whole construction of the hyper-tree is deterministic and a signature is entirely dependent of both the message and the secret key. This characteristic leads to the fact that no OTS can sign distinct messages, thus ensuring the security of the scheme.

⁶ Precisely, the signature contains a WOTS signature from which one can recover f_i .

In the following we present a fault injection attack allowing an attacker to recover the signature of two different messages with the same WOTS key.

Let us denote $\mathbf{sk} = (s_1, s_2, \dots, s_\ell)$ the targeted WOTS secret key corresponding to f_{d-1} and δ the Merkle tree root authenticated by it. We ask for the signature of a message \mathbf{m} about which we suppose, without loss of generality, that at the last step it requires signing δ . We note δ 's WOTS signature $\sigma_{d-1} = (\sigma_{d-1,1}, \dots, \sigma_{d-1,\ell})$. We receive the overall signature

$$\sigma = (\text{idx}, R_1, \sigma_H, \sigma_0, A(f_0), \dots, \sigma_{d-1}, A(f_{d-1})).$$

In the next step, we will ask again the signature of the same message \mathbf{m} . As the algorithm is entirely deterministic, the resulting signature should be the same as σ . However we will perturb the operations done in the computation of the authentication path $A(f_{d-2})$. This perturbation will result in the computation of a Merkle tree root δ^* distinct from δ . The resulting WOTS signature σ_{d-1}^* of δ^* gives the attacker the possibility to mount the grafting attack as shown previously. An overview of the fault can be seen in Fig. 4.

A nice feature of the fault model is that it is a very weak one. Indeed it verifies the following properties:

- only a single fault is needed per signature as a single fault in the computation of the Merkle tree of the penultimate layer fulfills the required modification;
- the fault is very permissive as we do not use the actual value of the faulted variable: we need the variable to change but do not need to know the actual value of the change;
- the fault can be done in a wide time period. Indeed, since the verification algorithm uses $A(f_{d-2})$ to compute δ^* , this authentication path must be faulted: otherwise, the attacker would deduce from it the legitimate root δ instead of the faulted one δ^* . This implies that one cannot directly fault the nodes which computations are redone by the signature verifier, but faulting all the other nodes will lead to a successful attack. In other words, one can fault any node “below” the authentication path, whereas it is not of interest for our purposes to fault any node “above”. In practice, it means that, in O-SPHINCS 33 227 hash computations may eventually be the target of the fault while 273 352 hash computations are available as targets in GRAVITY-SPHINCS with parameters given in Table 1.

These numbers stands for roughly 6% of the whole O-SPHINCS computation and 18% of the whole GRAVITY-SPHINCS⁷.

Remark 2. The faulted overall SPHINCS signatures produced by this attack are valid. Indeed, σ_{d-1}^* is the valid signature of δ^* , computed from $A(f_{d-2})^*$, which is given in the overall signature σ^* . Moreover, all other elements of σ^* are correct. Thus σ^* is accepted by the verification algorithm.

⁷ If the top layer of GRAVITY-SPHINCS is not cached, this percentage falls drastically but GRAVITY-SPHINCS also becomes very slow for these parameters, requiring about 2^{30} hashes per signature.

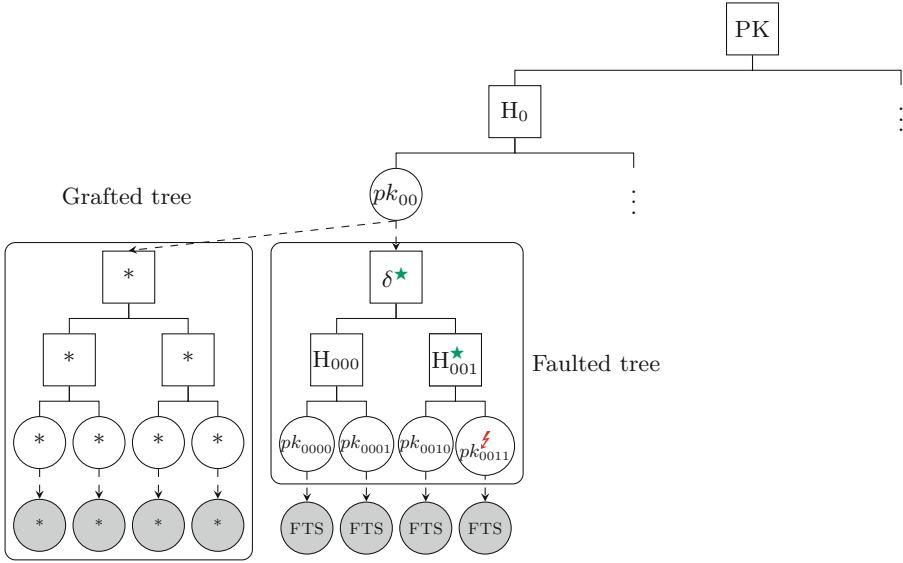


Fig. 4. Principle of our attack with a SPHINCS hyper-tree of height 4. Fault injection is denoted by a lightning and the affected elements of the faulted Merkle tree by a star. We note that the public key PK is not affected. The part of the hyper-tree which is below vertical dots is irrelevant to our attack.

3.3 Compromise Between Faulted Signatures and Computational Power

We have seen that one can achieve universal forgery on the existing schemes of the SPHINCS family at the price of one faulted signature. However, each of the signatures forged comes at a non-negligible computational cost as one needs to try about $1/p_w \approx 2^{34}$ values for the penultimate Merkle tree root (see Sect. 3.1) to be able to use the capacity on WOTS. In this section we provide trade-offs between the number of faulted signatures allowed to the attacker and the computational cost needed to forge a signature.

3.3.1 Total Break on WOTS

First, we estimate how many faulted signatures are necessary to recover an entire WOTS private key.

It is safe to model H as a random oracle. As a consequence of this hypothesis, when the computation of δ is faulted, δ^* takes a uniformly random value between 0 and $2^\lambda - 1$. Each b_i for $1 \leq i \leq \ell_1$ is therefore 0 with probability $1/w$.

Then the checksum C follows the law of a sum of $\ell_1 = 64$ random variables (r.v.) following the uniform law over $\llbracket 0, w - 1 \rrbracket$ which, thanks to the central limit theorem, we shall approximate by a normal law with parameters $\mu = \ell_1(w - 1)/2$ and $\sigma^2 = \ell_1(w^2 - 1)/12$.

Let us write C in base w in the big-endian convention. Then $b_i = 0$ for $\ell_1 < i \leq \ell$ means $C \bmod w^j < w^{j-1}$ with $j = i - \ell_1$. This event has probability:

$$\mathbb{P}(C \bmod w^j < w^{j-1}) = \sum_{q=0}^{\lfloor \ell_1(w-1)/w^j \rfloor} \sum_{z=qw^j}^{qw^j+w^{j-1}-1} \mathbb{P}(C = z),$$

with $\mathbb{P}(C = z) = \rho_{\mu,\sigma}(z) / \sum_{n \in \mathbb{Z}} \rho_{\mu,\sigma}(n)$, where $\rho_{\mu,\sigma}(x) = \exp(-(x-\mu)^2/(2\sigma^2))$.

We obtain $\mathbb{P}(b_{65} = 0) \approx 1/w$, $\mathbb{P}(b_{66} = 0) \approx 0.098$, $\mathbb{P}(b_{67} = 0) \approx 2^{-30.7}$. This last value calls for a remark. It means that we have to ask for $2^{30.7}$ signatures in average to get s_{67} , which is a lot, but, in the same time, we need s_{67} to sign a root with probability $2^{-30.7}$. So, with respect to s_{67} , we can only look for $H(s_{67})$. Given the very high probability of finding this value ($\mathbb{P}(b_{67} = 1) \approx 0.80$), we shall suppose that the average number of signatures required to recover s_1, \dots, s_{66} is high enough to find $H(s_{67})$ with overwhelming probability, and therefore we do not care about s_{67} anymore. We will later see that this is verified in practice.

Finally, we rely on the values of $\mathbb{P}(b_{65} = 0)$ and $\mathbb{P}(b_{66} = 0)$ to justify this approximation which will be done hereafter: b_1, b_2, \dots, b_{66} are viewed as 66 uniform deviate in $\llbracket 0, w-1 \rrbracket$.

Let us now consider the number of signatures required on average to carry out the attack. Let X be the random variable which models the number of requested signatures to find \mathbf{sk} (except s_{67}). Our problem then boils down to computing $\mathbb{E}(X)$.

Let $\{\sigma^{(1)*}, \sigma^{(2)*}, \dots, \sigma^{(n)*}\}$ be the set of the n requested faulted signatures at a certain point in the attack. We define: $V_j^{(n)} := \left\{ \sigma_{d-1,j}^{(i)*} \mid 1 \leq i \leq n \right\}$, that is, the set of values taken by the j th coordinate of all received $\sigma_{d-1}^{(i)*}$'s, when the attacker has gathered n faulted signatures. We also define the event $B_n := \{\exists 1 \leq j < \ell \text{ s.t. } s_j \notin V_j^{(n)}\}$. It can be shown that $\mathbb{P}(X = n) = \mathbb{P}(B_{n-1} \cap \overline{B_n})$ and that it leads to:

$$\mathbb{P}(X = n) = \mathbb{P}(\overline{B_n}) - \mathbb{P}(\overline{B_{n-1}}). \quad (2)$$

Since the coordinates of σ_{d-1}^* are pairwise independent and follow the same uniform law over $\llbracket 0, w-1 \rrbracket$ by assumption, we have:

$$\mathbb{P}(\overline{B_n}) = \prod_{j=1}^{\ell-1} \mathbb{P}(s_j \in V_j^{(n)}) = \left(1 - \left(\frac{w-1}{w} \right)^n \right)^{\ell-1}. \quad (3)$$

Composing Eqs. 2 and 3 with the parameters set in the SPHINCS schemes, we estimate that $\mathbb{E}(X) \approx 74.5$. Note that this leads to a probability to find $H(s_{67})$ greater than $1 - 2^{-170}$, which is indeed more than enough to do so.

One whole WOTS private key can thus be recovered querying 74.5 faulted signatures on average for the parameters used in the SPHINCS schemes.

3.3.2 Trade-Offs

We have seen that the attack can be mounted with only one faulted signature, with a non-negligible computational cost needed to forge every signatures, or that an attacker can make about 75 faulted signatures to ensure a free selection of the Merkle tree root. We now investigate the various trade-offs we can obtain by increasing step by step the number of faulted messages available to the attacker.

For this purpose, we extend Hülsing and Groot Bruinderink’s [GBH16] reasoning. Let us denote by δ the root which WOTS signature we want to forge, and by $\delta^{(1)}, \delta^{(2)}, \dots, \delta^{(n)}$ n uniformly random roots for which we have valid signatures by the same WOTS private key. Let $\mathbf{b} = (b_1, \dots, b_\ell)$ be the b -vector of δ and $\mathbf{b}^{(i)} = (b_1^{(i)}, \dots, b_\ell^{(i)})$ be the b -vector of $\delta^{(i)}$ for all $1 \leq i \leq n$. Then we can forge a valid WOTS signature for δ if and only if the following expression is true:

$$\bigwedge_{j=1}^{\ell} \bigvee_{i=1}^n \{b_j \geq b_j^{(i)}\}.$$

In order to estimate the probability of this event, we make the assumption that coordinates of the b -vector of a uniformly random word are pairwise independent and uniformly distributed in $\llbracket 0, w-1 \rrbracket$. If this assumption is clearly true for the first ℓ_1 coordinates, it is clearly not for the last ℓ_2 ones. However, we shall see later that the resulting theoretical probabilities are very close to probabilities obtained by simulations. Thus we work with this assumption for the sake of simplicity.

Moreover, we make the assumption that random roots $\delta^{(i)}$ are pairwise independent, *i.e.* that corresponding b -vectors are pairwise independent. By coordinates independence assumption:

$$\mathbb{P}\left(\bigwedge_{j=1}^{\ell} \bigvee_{i=1}^n \{b_j \geq b_j^{(i)}\}\right) = \mathbb{P}\left(\bigvee_{i=1}^n \{b_j \geq b_j^{(i)}\}\right)^\ell = \left(1 - \mathbb{P}(b_j < b_j^{(1)})^n\right)^\ell = \left(1 - \frac{1}{w^{n+1}} \sum_{a=0}^{w-1} a^n\right)^\ell, \quad (4)$$

the second equality coming from b -vectors independence assumption and the last one by identical distribution assumption. Table 2 presents the average number of roots to try before finding one whose signature can be forged, based on the number of faulted signatures the attacker has – note that the attacker is supposed to have the legitimate signature in addition to the faulted ones. The numbers are obtained from Eq. 4, and are matched by experiments.

We can observe that the computational complexity of the forgery of a message m is essentially the sum of the complexities of three operations:

Table 2. Number of grafted trees to generate randomly before finding one that can be signed by the faulted OTS, in function of the number of faulted signatures. We note that for 1 faulted signature, this number is $1/p_w$.

Faulted signatures	1	2	3	4	5	10	20
Number of trees to try	$2^{34.9}$	$2^{24.0}$	$2^{18.0}$	$2^{14.2}$	$2^{11.7}$	$2^{5.5}$	$2^{2.0}$

- the number of seeds to try to assign a satisfying index to the message – from 1 for O-SPHINCS to 2^{20} for GRAVITY-SPHINCS;
- the number of Merkle tree root values to try before being able to forge the WOTS signature – depending on the number of faulted signatures, see Table 2;
- the complexity of the signature⁸.

With this observation, we can state that only 3 faulted messages are needed to provide universal forgery against GRAVITY-SPHINCS at the cost of about 2^{20} hash computations.

3.4 Countermeasures

Generic countermeasures such as making the signature computation redundant can complicate our attack, but they may incur a significant overhead (for redundancy, a factor 2 in time and space). Indeed, a simple verification of the signature would not be efficient in our case as the attack provides valid signatures. Moreover, only a small part of the execution will be faulted and thus the redundancy must be checked for each of the roots of a Merkle sub-tree. However redundant computation is an efficient way to significantly constrain the attacker to a more powerful model as the fault should be exactly replicable on both executions. As generic countermeasures are well documented, our discussion will focus on the countermeasures that are specific to the SPHINCS framework.

In [MKAA16] the authors propose a specific recomputation designed to avoid faults in Merkle trees, called Recomputing with Swapped Nodes (RESN). Whereas the countermeasure provides efficient security at an acceptable overhead by lightly pipelining the circuit, it does not cover the Goldreich construction. The main impact to our attack additionally from classical recomputation methods is that it limits the fault to targeting only the computation of the root of the Merkle tree as any other faulted hash computation would be detected by the RESN. We note that, in this case, the faulted signature will not be a valid one anymore and the result could be verified with an additional overcost.

A naïve way to protect SPHINCS against our attack would be to compute the index of the FTS from public values instead of secret ones. Indeed, if one computes the index from the message and the public key, the attacker is not able anymore to choose the index of the message to sign.

However, the cure would be worse than the disease. Indeed, while our attack would be thwarted by this modification, a malicious user would now be able to provoke multi-collisions on the index idx of the FTS by trying several messages.

In the schemes studied, the number of FTS leaves is upper bounded by 2^{64} . This implies that given a fixed value for the index idx , an attacker can find k messages leading to this index with a computational effort about $k \times 2^{64}$. Therefore, this modification would lead to universal forgeability of the targeted schemes *without any fault*.

⁸ The complexity of the forged signature can be slightly lowered because the attacker does not need to compute valid values for the authentication path and can simply generate random values.

An efficient countermeasure would be to somehow link the different layers of the hyper-tree so that a fault in the computation of the tree would result in a non valid signature, *i.e.* a root value distinct from the public key. A simple check of the validity before returning the signature would prevent any fault attack. However, in order to link these layers, one cannot compute the OTS keys only from its index and the secret key, so the whole hyper-tree would have to be recomputed for each signature, ensuing a huge overhead in signing time.

4 Conclusion and Open Questions

In this paper we propose the first fault attack against signature schemes of the SPHINCS family. After an initial cost of a single faulted message, it allows to forge signatures for any message at an (offline) cost of 2^{34} hashes per message.

We proposed several trade-offs to lower this computational cost while slightly increasing the number of faulted messages. For any of the targeted schemes, we can forge any message at a cost of about 2^{20} hashes functions knowing only 3 faulted messages. Moreover, the fault model is very permissive.

While our attack can be thwarted by generic (but possibly costly) countermeasures against fault attacks, we did not find any specific countermeasure.

As demonstrated by this work, the deterministic nature of several hash-based signatures and their internal use of OTS can be a weakness against fault attacks. On the defensive side, an interesting line of work would be to propose hash-based constructions which offer some innate resilience against fault attacks.

On the offensive side, a natural extension of this work would be to implement the proposed fault attack in practice. Our attack target the SPHINCS framework, but it would be interesting to extend it to other multi-tree constructions such as multi-tree XMSS or GMSS. One could also devise an alternative way (other than fault injection) to recover two distinct WOTS signatures for the same key, which would allow to apply our grafting attack. We leave this for future work.

Acknowledgements. We would like to thank the anonymous PQCrypto reviewers for their helpful comments. We also thank Andreas Hülsing, whose insightful advices helped us make our attack simpler, more generic and more powerful. Finally, we acknowledge the support of the French *Programme d'Investissement d'Avenir* under national project RISQ.

References

- [AE17a] Aumasson, J.-P., Endignoux, G.: Clarifying the subset-resilience problem. Cryptology ePrint Archive, Report 2017/909 (2017). <https://eprint.iacr.org/2017/909>
- [AE17b] Aumasson, J.-P., Endignoux, G.: Improving stateless hash-based signatures. Cryptology ePrint Archive, Report 2017/933 (2017). <https://eprint.iacr.org/2017/933>
- [BBK16] Bindel, N., Buchmann, J.A., Krämer, J.: Lattice-Based Signature Schemes and Their Sensitivity to Fault Attacks (2016)

- [BDE+17] Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.-L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P.: SPHINCS+ (2017). <https://sphincs.org/>
- [BDH11] Buchmann, J., Dahmen, E., Hülsing, A.: XMSS - a practical forward secure signature scheme based on minimal security assumptions. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 117–129. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_8
- [BDK+07] Buchmann, J., Dahmen, E., Klintsevich, E., Okeya, K., Vuillaume, C.: Merkle signatures with virtually unlimited signature capacity. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 31–45. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72738-5_3
- [BDL97] Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_4
- [BDS08] Buchmann, J., Dahmen, E., Schneider, M.: Merkle tree traversal revisited. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 63–78. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88403-3_5
- [BG15] Blömer, J., Günther, P.: Singular curve point decompression attack. In: FDTC, pp. 71–84. IEEE Computer Society (2015)
- [BGS15] Bagheri, N., Ghaedi, N., Sanadhya, S.K.: Differential fault analysis of SHA-3. In: Biryukov, A., Goyal, V. (eds.) INDOCRYPT 2015. LNCS, vol. 9462, pp. 253–269. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26617-6_14
- [BHH+15] 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
- [EFGT16] Espitau, T., Fouque, P.-A., Gérard, B., Tibouchi, M.: Loop-abort faults on lattice-based fiat-shamir and hash-and-sign signatures. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 140–158. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_8
- [GBH16] Bruinderink, L.G., Hülsing, A.: “Oops, i did it again” - security of one-time signatures under two-message attacks. IACR Cryptology ePrint Archive (2016). <http://eprint.iacr.org/2016/1042>
- [Gol86] Goldreich, O.: Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 104–110. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_8
- [GW17] Gélin, A., Wesolowski, B.: Loop-abort faults on supersingular isogeny cryptosystems. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 93–106. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_6
- [HBB12] Hülsing, A., Busold, C., Buchmann, J.: Forward secure signatures on smart cards. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 66–80. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_5

- [HH11] Hemme, L., Hoffmann, L.: Differential fault analysis on the SHA1 compression function. In: 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, 29 September 2011, pp. 54–62 (2011)
- [HRB13] Hülsing, A., Rausch, L., Buchmann, J.: Optimal parameters for XMSS^{MT}. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CDARES 2013. LNCS, vol. 8128, pp. 194–208. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40588-4_14
- [HRS16] Hülsing, A., Rijneveld, J., Schwabe, P.: ARMed SPHINCS - computing a 41 KB signature in 16 KB of RAM. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 446–470. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_17
- [Lam79] Lamport, L.: Constructing digital signatures from a one way function. Technical report SRI-CSL-98, SRI International Computer Science Laboratory (1979)
- [Mer90] Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_21
- [MKAA16] Mozaffari-Kermani, M., Azarderakhsh, R., Aghaie, A.: Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarked on ASIC. *ACM Trans. Embed. Comput. Syst.* **16**(2), 59 (2016)
- [NIS16] NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016). <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography>
- [RED+08] Rohde, S., Eisenbarth, T., Dahmen, E., Buchmann, J., Paar, C.: Fast hash-based signatures on constrained devices. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 104–117. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85893-5_8
- [Rom90] Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: STOC, pp. 387–394. ACM (1990)
- [Son14] Song, F.: A note on quantum security for post-quantum cryptography. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 246–265. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_15
- [Ti17] Ti, Y.B.: Fault attack on supersingular isogeny cryptosystems. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 107–122. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_7



Post-quantum Security of the Sponge Construction

Jan Czajkowski¹(✉) , Leon Groot Bruinderink²(✉) , Andreas Hülsing²(✉) ,
Christian Schaffner¹(✉) , and Dominique Unruh³(✉) 

¹ QuSoft, University of Amsterdam, Amsterdam, The Netherlands

{j.czajkowski,c.schaffner}@uva.nl

² TU Eindhoven, Eindhoven, The Netherlands

l.groot.bruinderink@tue.nl, andreas@huel sing.net

³ University of Tartu, Tartu, Estonia

unruh@ut.ee

Abstract. We investigate the post-quantum security of hash functions based on the sponge construction. A crucial property for hash functions in the post-quantum setting is the collapsing property (a strengthening of collision-resistance). We show that the sponge construction is collapsing (and in consequence quantum collision-resistant) under suitable assumptions about the underlying block function. In particular, if the block function is a random function or a (non-invertible) random permutation, the sponge construction is collapsing. We also give a quantum algorithm for finding collisions in an arbitrary function. For the sponge construction, the algorithm complexity asymptotically matches the complexity implied by collision resistance.

Keywords: Sponge construction · QROM · Collapsing
Collision resistance · Quantum algorithms

1 Introduction

Cryptographic hash functions are one of the central primitives in cryptography. They are used virtually everywhere: As cryptographically secure checksums to verify integrity of software or data packages, as building block in security protocols, including TLS, SSH, IPSEC, as part of any efficient variable-input-length signature scheme, to build full-fledged hash-based signature schemes, in transformations for CCA-secure encryption, and many more.

This work was supported in part by the Commission of the European Communities through the Horizon 2020 program under project number 645622 PQCRYPTO. CS and JC are supported by a NWO VIDI grant (Project No. 639.022.519). DU was supported by institutional research funding IUT2-1 of the Estonian Ministry of Education and Research, and by the Estonian Centre of Excellence in IT (EXCITE) funded by the ERDF, and the Estonian ICT program 2011–2015 (3.2.1201.13-0022).

While all widely deployed public-key cryptography is threatened by the rise of quantum computers, hash functions are believed to be only mildly affected. The reason for this is twofold. On the one hand, generic quantum attacks achieve at most a square-root speed up compared to their pre-quantum counterparts and can be proven asymptotically optimal [8, 13, 20]. On the other hand, there do not exist any dedicated quantum attacks on any specific hash function that perform better than the generic quantum attacks (except, of course, for hash functions based on number theory like, e.g., VSH [9]).

One of the most important properties of a hash function H is collision-resistance. That is, it is infeasible to find $x \neq x'$ with $H(x) = H(x')$. Intuitively, collision-resistance guarantees some kind of computational injectivity – given $H(x)$, the value x is effectively determined. Of course, information-theoretically, x is not determined, but in many situations, we can treat the preimage x as unique, because we will never see another value with the same hash. For example, collision-resistant hashes can be used to extend the message space of signature schemes (by signing the hash of the message), or to create commitment schemes (e.g., sending $H(x||r)$ for random r commits us to x ; the sender cannot change his mind about x because he cannot find another preimage).

In the post-quantum setting,¹ however, it was shown by Unruh [18] that collision-resistance is weaker than expected: For example, the commitment scheme sketched in the previous paragraph is not binding: it is possible for an attacker to send a hash h , then to be given a value x , and then to send a random value r such that $h = H(x||r)$, thus opening the commitment to any desired value – even if H is collision-resistant against quantum adversaries.² This contradicts the intuitive requirement that $H(x)$ determines x .

Fortunately, Unruh [18] also presented a strengthened security definition for post-quantum secure hash functions: collapsing hash functions. Roughly speaking, a hash function is collapsing if, given a superposition of values m , measuring $H(m)$ has the same effect as measuring m (at least from the point of view of a computationally limited observer). Collapsing hash functions serve as a drop-in replacement for collision-resistant ones in the post-quantum setting: Unruh showed that several natural classical commitment schemes (namely the scheme sketched above, and the statistically-hiding schemes from [12]) become post-quantum secure when using a collapsing hash function instead of a collision-resistant one. The collapsing property also directly implies collision-resistance.

In light of these results, it is desirable to find hash functions that are collapsing. Unruh [18] showed that the random oracle is collapsing. (That is, a hash function $H(x) := \mathcal{O}(x)$ is collapsing when \mathcal{O} is a random oracle.) However, this has little relevance for real-world hash functions: A practical hash function

¹ We mean a situation in which the protocols and primitives that are studied are classical, but the attacker can perform quantum computations.

² More precisely, [18] shows that relative to certain oracles, a collision-resistant hash function exists that allows such attacks. In particular, this means that there cannot be a relativizing proof that the commitment scheme is binding assuming a collision-resistant hash function.

is typically constructed by iteratively applying some elementary building block (e.g., a “compression function”) in order to hash large messages. So even if we are willing to model the elementary building block as a random oracle, the overall hash function construction should arguably not be modeled as a random oracle.³

For hash functions based on the Merkle-Damgård (MD) construction (such as SHA2 [16]), Unruh [19] showed: If the compression function is collapsing, so is the hash function resulting from the MD construction. In particular, if we model the compression function as a random oracle (as is commonly done in the analysis of practical hash functions), we have that hash functions based on the MD construction are collapsing (and thus suitable for use in a post-quantum setting).

However, not all hash functions are constructed using MD. Another popular construction is the sponge construction [4], underlying for example the current international hash function standard SHA3 [17], but also other hash functions such as Quark [2], Photon [11], Spongent [6], and Gluon [3]. The sponge construction builds a hash function H from a block function⁴ \mathbf{f} . In the classical setting, we know that the sponge construction is collision-resistant if the block function \mathbf{f} is modeled as a random oracle, or a random permutation, or an invertible random permutation [5].⁵ However, their proof does not carry over to the post-quantum setting: their proof relies on the fact that queries performed by the adversary to the block function are classical (i.e., not in superposition between different values). As first argued in [7], random oracles and related objects should be modeled as functions that can be queried in superposition of different inputs. (Namely, with a real hash function, an adversary can use a quantum circuit implementing SHA3 and can thereby query the function in superposition. The adversary could evaluate the sponge on the uniform superposition over all messages of a certain length, possibly helping him to, e.g., find a collision.) Thus, we do not know whether the sponge construction (and thus hash functions like SHA3) is collapsing (or at least collision-resistant in the post-quantum setting).

Our contributions. In the present paper we tackle the question whether the sponge construction is collision-resistant and collapsing in the post-quantum setting. We show:

- If the block function \mathbf{f} is collision-resistant when restricted to the left and right half of its output and it is hard to find a zero-preimage of \mathbf{f} (restricted to the right half of its output), then the sponge construction is collision resistant.

³ For example, hash functions using the Merkle-Damgård construction are not well modeled as a random oracle. If we use $MAC(k, m) := H(k||m)$ as a message authentication code (MAC) with key k , we have that MAC is secure (unforgeable) when H is a random oracle, but easily broken when H is a hash function built using the Merkle-Damgård construction.

⁴ It is not called a compression function, since the domain and range of \mathbf{f} are identical.

⁵ [5] shows that the sponge construction is indifferentiable from a random oracle *in the classical setting*. Together with the fact that the random oracle is collision-resistant, collision-resistance of the sponge construction follows.

- If the block function \mathbf{f} is collapsing when restricted to the left and right half of its output, respectively, and if it is hard to find a zero-preimage of \mathbf{f} (restricted to the right half of its output), then the sponge construction is collapsing.
- If the block function \mathbf{f} is a random oracle or a random permutation, then the sponge construction is collapsing.
- We give a quantum algorithm for finding collisions in any function (given access to a random oracle), in particular in the sponge construction. The number of quantum queries to \mathbf{f} asymptotically matches our bounds for collision resistance.

It should be stressed that we *do not* show that the sponge construction is collapsing (or even collision-resistant) if the block function \mathbf{f} is an *efficiently invertible* random permutation. In this case, it is trivial to find zero-preimages by applying the inverse permutation to 0. This means that the present result cannot be directly used to show the security of, say, SHA3, because SHA3 uses an efficiently invertible permutation as block function. Our results apply to hash functions where the block function is not (efficiently) invertible, e.g., Gluon [3]. It seems that this limitation is just a residue of our technique.

Organization. In Sect. 2 (“Collapsing hash functions”), we recall the definition of collapsing hash functions and some important properties of that definition. In Sect. 3 (“The sponge construction”) we recall the sponge construction. In Sect. 4 (“Collision-resistance of the sponge construction”) we first show the collision resistance of the sponge construction. Then in Sect. 5 (“Sponges are collapsing”), we present our main result – that the sponge construction is collapsing. In Sect. 6 (“Quantum Attack”) we present a quantum algorithm for finding collisions that uses a random oracle. Additional details, including preliminaries and full proofs are given in the full version [10].

2 Collapsing Hash Functions

In this section, we recall the notion of collapsing hash functions H from [18]. We describe both the underlying intuition, as well as the formal definitions.

A *hash function* is a function $H^{\mathcal{O}} : X \rightarrow Y$ for some range X and domain Y . (Typically, Y consists of fixed length bitstrings, and X consists of fixed length bitstrings or $\{0, 1\}^*$.) H can depend on an oracle \mathcal{O} . (Typically, \mathcal{O} will be a random function, a random permutation, or simply be missing if we are in the standard model. Unless specified otherwise, we make no assumptions about the distribution of \mathcal{O} .)

As mentioned in the introduction, intuitively, we wish that $H(m)$ uniquely identifies m in some sense. In the classical setting, this naturally leads to the requirement that it is hard to find $m \neq m'$ with $H(m) = H(m')$. Then we can treat $H(m)$ as if it had only a single preimage (even though, of course, a compressing H will have many preimages, we just cannot find them). In the quantum setting, there is another interpretation of the requirement that $H(m)$ identifies m . Namely, if we are given a register M that contains a superposition of many values m , then measuring $H(m)$ on that register should – intuitively –

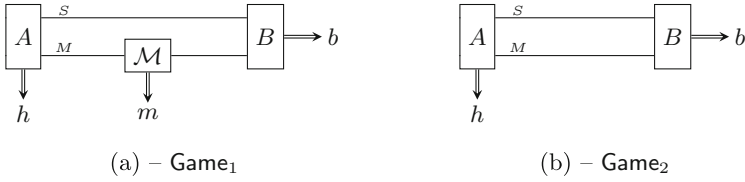


Fig. 1. Games from the definition of collapsing hash functions. \mathcal{M} represents a measurement in the computational basis. (A, B) is assumed to satisfy the property that \mathcal{M} always returns m with $H(m) = h$. A function is collapsing if the probability of $b = 1$ is negligibly close in both games.

fully determine m . That is, the effect on the register M should be the same, no matter whether we measure just the hash $H(m)$ or the whole message m . One can see that for any compressing function H , it is impossible that measuring $H(m)$ and m has information-theoretically the same effect on the state.⁶ However, what we can hope for is that for a computationally limited adversary, the two situations are indistinguishable. In other words, we require that no quantum-polynomial-time adversary can distinguish whether we measure $H(m)$ -measurements by m -measurements and vice versa.

We can slightly simplify this condition if we require that the register M already contains a superposition of values m that all have the same hash $H(m)$. In this case, measuring $H(m)$ has no effect on the state, so we can state the requirement as: If M contains a superposition of messages m with the same $H(m) = h$, then no quantum-polynomial-time adversary can distinguish whether we measure M in the computational basis, or whether we do not measure it at all.

Or slightly more formally: We let the adversary A produce a register M and a hash value h (subject to the promise that measuring M would lead to an m with $H(m) = h$). The adversary additionally keeps an internal state in register S . Then we either measure M in the computational basis (Game₁, depicted in Fig. 1(a)), or we do not perform any such measurement (Game₂, depicted in Fig. 1(b)). Finally, we give registers S (the internal state) and M (the potentially measured message register) to the adversary’s second part B . We call H *collapsing* if no quantum-polynomial-time (A, B) can distinguish Game₁ and Game₂.

This is formalized by the following definition:

Definition 1 (Collapsing [18]). For algorithms A, B , consider the following games:

$$\begin{aligned} \text{Game}_1 : & \quad (S, M, h) \leftarrow A^\mathcal{O}(), \quad m \leftarrow \mathcal{M}(M), \quad b \leftarrow B^\mathcal{O}(S, M) \\ \text{Game}_2 : & \quad (S, M, h) \leftarrow A^\mathcal{O}(), \quad b \leftarrow B^\mathcal{O}(S, M) \end{aligned}$$

⁶ E.g., M could contain $\sum_m 2^{-|m|/2} |m\rangle$. Then measuring $H(m)$ will lead to the state $\sum_{m \text{ s.t. } H(m)=h} \frac{1}{\sqrt{|H^{-1}(h)|}} |m\rangle$ which is almost orthogonal for large $|H^{-1}(h)|$ to the state $|m\rangle$ we get when measuring m .

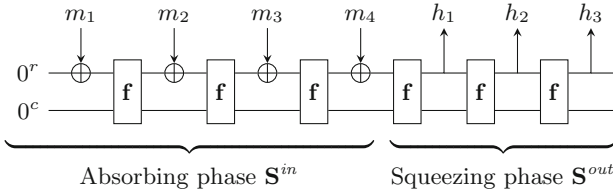


Fig. 2. The sponge construction S with a four block input $m_1||m_2||m_3||m_4$ and a three block output $h_1||h_2||h_3$. The application of the padding function is not depicted (we assume $m_1||m_2||m_3||m_4 = pad(m)$).

Here S, M are quantum registers. $\mathcal{M}(M)$ is a measurement of M in the computational basis.

For a set \mathbf{m} , we call an adversary (A, B) valid on \mathbf{m} for $H^\mathcal{O}$ iff $\Pr[H^\mathcal{O}(m) = h \wedge m \in \mathbf{m}] = 1$ when we run $(S, M, h) \leftarrow A^\mathcal{O}()$ and measure M in the computational basis as m . If we omit “on \mathbf{m} ”, we assume \mathbf{m} to be the domain of $H^\mathcal{O}$.

A function H is collapsing (on \mathbf{m}) iff for any quantum-polynomial-time adversary (A, B) that is valid for $H^\mathcal{O}$ (on \mathbf{m}), $|\Pr[b = 1 : \text{Game}_1] - \Pr[b = 1 : \text{Game}_2]|$ is negligible.

The definition follows [18], except that we made the oracle \mathcal{O} explicit (which was implicit in [18]).

Miscellaneous facts. The following properties of collapsing hash functions will be useful throughout this paper. They are immediate consequences of their concrete-security variants as shown in Sect. 3.1 of the full version [10].

Lemma 2. If $H^\mathcal{O}$ is injective, then $H^\mathcal{O}$ is collapsing.

Theorem 3. If $\mathcal{O} : \{0, 1\}^e \rightarrow \{0, 1\}^d$ is a random function with superlogarithmic d (in the security parameter), then $H^\mathcal{O} := \mathcal{O}$ is collapsing.

Lemma 4. If $G^\mathcal{O} \circ H^\mathcal{O}$ is collapsing, and $G^\mathcal{O}$ is quantum-polynomial-time computable, then $H^\mathcal{O}$ is collapsing.

Lemma 5. If $G^\mathcal{O}$ and $H^\mathcal{O}$ are collapsing, and $H^\mathcal{O}$ is quantum-polynomial-time computable, then $G^\mathcal{O} \circ H^\mathcal{O}$ is collapsing.

3 The Sponge Construction

In this section, we review the sponge construction introduced by [4]. The sponge construction has two internal parameters r and c called the *rate* and the *capacity*, respectively. The internal state has $r+c$ bits. We refer to the first part of the state as the left state, and to the second part of the state as the right state. Underlying the sponge construction is a block function f that inputs and outputs $r+c$ bits.

To hash a message m , the message is first padded to a non-zero multiple of the rate r . That is, we use some injective *padding function* pad to get $k \geq 1$ message blocks $m_1 \parallel \dots \parallel m_k = pad(m)$.⁷ Then we XOR m_1 to the left state, apply \mathbf{f} to the (whole) state, XOR m_2 to the left state, apply \mathbf{f} to the state, \dots , apply \mathbf{f} to the state, XOR m_k to the left state. The steps performed so far are referred to as the *absorbing phase* (denoted in this paper with \mathbf{S}^{in}). Now we start with the *squeezing phase* \mathbf{S}^{out} : We apply \mathbf{f} to the state, read the left state as h_1 , apply \mathbf{f} to the state, read the left state as h_2 , \dots . We continue to do so until $h_1 \parallel h_2 \parallel \dots$ contains $\geq n$ bits (where n is a parameter specifying the desired output length), and return the first n bits of $h_1 \parallel h_2 \parallel \dots$. The whole process described here (padding, absorbing phase, squeezing phase) is the sponge construction, referred to as \mathbf{S} in this paper. Note that the use of the terms absorbing and squeezing phase in this paper slightly differ from the description in [4]: In this paper, we end the absorbing phase just before the last application of \mathbf{f} , whilst the original sponge paper includes that application of \mathbf{f} in the absorbing phase. The separation we use helps to simplify the proofs in later sections. The resulting sponge construction is the same as in [4], though. The sponge construction is illustrated in Fig. 2 for the special case of $k = 4$ and $n = 3r$ (four input blocks and three output blocks). The following definition makes the above explanation precise:

Definition 6 (Sponge construction). Fix integers $c > 0$ (the capacity) and $r > 0$ (the rate), and $n > 0$ (the output length). Fix $\mathbf{f} : \{0, 1\}^{r+c} \rightarrow \{0, 1\}^{r+c}$ (the block function) and $pad : \{0, 1\}^* \rightarrow (\{0, 1\}^r)^+$ (where $\{0, 1\}^r$ is the set of bit-strings consisting of r -bit blocks).

For $m_1, \dots, m_k \in \{0, 1\}^r$, let

$$\begin{aligned} \mathbf{S}_{c,r,\mathbf{f}}^{in}(m_1 \parallel \dots \parallel m_k) &:= \mathbf{f}(\mathbf{S}_{c,r,\mathbf{f}}^{in}(m_1 \parallel \dots \parallel m_{k-1})) \oplus (m_k \parallel 0^c) \\ \mathbf{S}_{c,r,\mathbf{f}}^{in}(m_1) &:= m_1 \parallel 0^c \end{aligned}$$

(We call \mathbf{S}^{in} the absorbing phase.)

For $s \in \{0, 1\}^{r+c}$, let

$$\mathbf{S}_{c,r,\mathbf{f},n}^{out}(s) = \begin{cases} s' \parallel \mathbf{S}_{c,r,\mathbf{f},n-|s'|}^{out}(\mathbf{f}(s)) & (n > 0) \\ \text{empty word} & (n = 0) \end{cases}$$

where s' consists of the first $\min\{n, r\}$ bits of $\mathbf{f}(s)$. (We call \mathbf{S}^{out} the squeezing phase.)

Let $\mathbf{S}_{c,r,\mathbf{f},pad,n} := \mathbf{S}_{c,r,\mathbf{f},n}^{out} \circ \mathbf{S}_{c,r,\mathbf{f}}^{in} \circ pad$. We call $\mathbf{S}_{c,r,\mathbf{f},pad,n}$ the sponge construction.

Usually, c, r, \mathbf{f}, pad, n will be clear from the context. Then we omit them and simply write \mathbf{S}^{in} , \mathbf{S}^{out} , and \mathbf{S} .

⁷ The original construction requires that the last block of $pad(m)$ is non-zero, this is important for other properties than collision-resistance/collapsing. In this work, we do not put any such requirement on pad . We do, however, assume that pad outputs at least one block.

Notation: The sponge construction operates on a state of size $r + c$, and we will often need to refer to the two halves of that state separately: For any $s \in \{0, 1\}^{r+c}$, let s^{left} denote the first r bits of s , and let s^{right} refer to the last c bits of s . If f is a function with $r + c$ bit output, then we write f^{left} for the function defined by $f^{\text{left}}(x) := f(x)^{\text{left}}$. And f^{right} analogously.

As the output of the sponge function can be smaller than the rate, i.e. $n \leq r$, we also define the function $\mathbf{f}^{\text{left}/n} : \{0, 1\}^{r+c} \rightarrow \{0, 1\}^{\min(n,r)}$, which is the function that outputs the first $\min(n, r)$ bits of \mathbf{f} . In particular, $\mathbf{f}^{\text{left}/n} := \mathbf{f}^{\text{left}}$ for $n \geq r$.

4 Collision-Resistance of the Sponge Construction

In this section we state our result concerning collision-resistance of the sponge construction. We motivate our statement with Lemma 8 connecting attacks on some features of the block function with collision-resistance of the overall construction. Those features are collision-resistance of $\mathbf{f}^{\text{right}}$, collision resistance of $\mathbf{f}^{\text{left}/n}$, and zero-preimage-resistance of $\mathbf{f}^{\text{right}}$.

The last notion is defined as follows: a function $f^{\mathcal{O}} : \{0, 1\}^d \rightarrow \{0, 1\}^e$ is *zero-preimage-resistant* iff for any quantum-polynomial-time adversary $A^{\mathcal{O}}$, we have that $\Pr[f^{\mathcal{O}}(x) = 0^e : x \leftarrow A^{\mathcal{O}}()]$ is negligible. Concrete security bounds are given in the full version [10].

Let us state the main result of this section.

Theorem 7. *Assume that $\mathbf{f}^{\text{right}}$ and $\mathbf{f}^{\text{left}/n}$ are collision resistant and $\mathbf{f}^{\text{right}}$ is zero-preimage resistant. Then $\mathbf{S}_{c,r,\mathbf{f},\text{pad},n}$ is collision-resistant.*

Proof sketch. We prove this theorem by a reduction to adversaries attacking the block function. Namely finding collisions in $\mathbf{f}^{\text{right}}$ or $\mathbf{f}^{\text{left}/n}$, or a zero-preimage under $\mathbf{f}^{\text{right}}$. This reduction is presented in Lemma 8. Knowing that every collision in \mathbf{S} results in breach in the security of $\mathbf{f}^{\text{right}}$ or $\mathbf{f}^{\text{left}/n}$, allows us to state the claim of the theorem. \square

Lemma 8. *Assume that pad is injective. There is a deterministic polynomial-time oracle algorithm A such that for any $m \neq \hat{m}$ with $\mathbf{S}(m) = \mathbf{S}(\hat{m})$, $A^{\mathbf{f}}(m, \hat{m})$, outputs one of the following:*

- (right, (s, \hat{s})) where (s, \hat{s}) is a collision of $\mathbf{f}^{\text{right}}$,
- (zero, s) where s is a zero-preimage of $\mathbf{f}^{\text{right}}$,
- or (left, (s, \hat{s})) where (s, \hat{s}) is a collision of $\mathbf{f}^{\text{left}/n}$.

Proof. A starts by computing the first right-state of the squeezing phase on input of the two colliding messages, i.e., it evaluates $\mathbf{f} \circ \mathbf{S}^{in} \circ \text{pad}$. We will denote the states traversed during this calculation by s_i and \hat{s}_i for m and \hat{m} , respectively. As our analysis starts with the final state of this computation and revisits the intermediate states in backwards direction, we denote by s_0 the final state, whose left part is output (for $n < r$ only the first n bits), by s_{-1} the state just before the last application of \mathbf{f} and so on. A figure including this notation is presented

later in Fig. 3. Using $p := \text{pad}(m)$ and $\hat{p} := \text{pad}(\hat{m})$, the intermediate states s_{-i} for $1 \leq i \leq |p| - 1$ are defined by $s_{-i} := \mathbf{f}(s_{-i-1}) \oplus p_{|p|+1-i} \| 0^c$, $s_0 := \mathbf{f}(s_{-1})$ and $s_{-|p|} := p_{|p|} \| 0^c$. As m and \hat{m} collide per assumption, we have $s_0^{\text{left}/n} = \hat{s}_0^{\text{left}/n}$.

1. Algorithm A first checks if s_{-1} or \hat{s}_{-1} are a preimage of 0^c , or form a collision under $\mathbf{f}^{\text{left}/n}$. If the right part of s_0 (or \hat{s}_0) is 0^c , s_{-1} (\hat{s}_{-1}) is a pre-image of 0^c under $\mathbf{f}^{\text{right}}$ and A outputs (zero, s_{-1}) ($(\text{zero}, \hat{s}_{-1})$, respectively). If $s_{-1} \neq \hat{s}_{-1}$, A outputs $(\text{left}, s_{-1}, \hat{s}_{-1})$. These two states form a collision under $\mathbf{f}^{\text{left}/n}$ because they are the inputs to the last \mathbf{f} in \mathbf{S} and $s_0^{\text{left}/n} = \hat{s}_0^{\text{left}/n}$. Otherwise, $s_{-1} = \hat{s}_{-1}$ and there are no preimages of zero.
2. If not done yet, $s_{-1} = \hat{s}_{-1}$ and A checks for a preimage of 0^c or a collision in $\mathbf{f}^{\text{right}}$. If $s_{-1}^{\text{right}} = 0^c$, A found a preimage of 0^c . This is true as if both messages ended here then $s_{-1} = \hat{s}_{-1}$ would imply that $p = \hat{p}$ (and so $m = \hat{m}$) which contradicts the assumptions of the lemma. Hence, at least one message must be longer. Assuming the longer message is m , A outputs (zero, s_{-2}) (or $(\text{zero}, \hat{s}_{-2})$ if it was \hat{m}).

Next the algorithm checks if $p_{-1} = \hat{p}_{-1}$, where we follow a similar notation for message blocks as for the states. The last block of the input is denoted by p_{-1} . If $p_{-1} \neq \hat{p}_{-1}$, A outputs $(\text{right}, s_{-2}, \hat{s}_{-2})$. This is a collision of $\mathbf{f}^{\text{right}}$ because $p_{-1} \neq \hat{p}_{-1}$ but $s_{-1} = \hat{s}_{-1}$. Thus $\mathbf{f}(s_{-2}) \neq \mathbf{f}(\hat{s}_{-2})$ which in turn implies $s_{-2} \neq \hat{s}_{-2}$ while $\mathbf{f}^{\text{right}}(s_{-2}) = \mathbf{f}^{\text{right}}(\hat{s}_{-2})$. We can be certain that there are at least two applications of \mathbf{f} both in $\mathbf{S}(m)$ and $\mathbf{S}(\hat{m})$ because the right half of $s_{-1} = \hat{s}_{-1}$ is not 0^c .

3. If $p_{-1} = \hat{p}_{-1}$ we end up in the same situation as before but now for $i = 2$. Namely we have that $s_{-2} = \hat{s}_{-2}$ and the algorithm performs the same checks as before but for a bigger i . Repeat Step 2 for all $2 \leq i \leq \min\{|p|, |\hat{p}|\}$.

If the iteration ends without success, this especially means that no collision was found but at least one message was fully processed. In this case A outputs a preimage of 0^c under $\mathbf{f}^{\text{right}}$. That is because no collisions means that all compared message blocks are the same but the two messages are different per assumption. Hence, they must have different lengths. With different length messages that traverse the same state values at the point of $i = \min\{|p|, |\hat{p}|\}$ the right part of both states is 0^c , so the algorithm will output $(\text{zero}, \hat{s}_{-|p|-1})$ (assuming $|p| < |\hat{p}|$). \square

5 Sponges are Collapsing

In this section, we show that the sponge construction is collapsing, under certain assumptions about the block function \mathbf{f} . We only state the qualitative results here, more precise statements with concrete security bounds will be given in the full version [10]. The results in this section hold for all distributions of the oracle \mathcal{O} (including the case that there is no oracle \mathcal{O}). The specific cases of random functions and random permutations are covered in Sect. 5.1. Since all adversaries (A, B, A', B', \dots) and the block function \mathbf{f} have oracle access to \mathcal{O}

throughout the section, we omit the oracle \mathcal{O} from our notation for increased readability (i.e., we write A, \mathbf{f} instead of $A^{\mathcal{O}}, \mathbf{f}^{\mathcal{O}}$). Throughout this section, we assume that $\mathbf{f}^{\mathcal{O}}$ can be computed in quantum-polynomial-time (given oracle access to \mathcal{O}).

We will analyze the sponge construction in three parts. First, we analyze the security of the absorbing phase \mathbf{S}^{in} , then we analyze the security of the squeezing phase \mathbf{S}^{out} , and finally we conclude security of the whole sponge \mathbf{S} , consisting of padding, absorbing, and squeezing.

First, we analyze the absorbing phase. For the absorbing phase (without padding or squeezing) to be collapsing, we will need two properties of \mathbf{f}^{right} :

- \mathbf{f}^{right} is collapsing. This is the main property required from the block function \mathbf{f} . If we would restrict \mathbf{S}^{in} to fixed length messages, then we could show the collapsing property of \mathbf{S}^{in} based on that property alone.
- \mathbf{f}^{right} is zero-preimage-resistant.

To see why we need this property, consider a block function \mathbf{f} where the adversary can find, e.g., $x, y \in \{0, 1\}^r$ with $f(x||0^c) = y||0^c$. Then we can see that $\mathbf{S}^{in}(x||y) = 0^{c+r}$, and thus $\mathbf{S}^{in}(x||y||z) = z||0^c = \mathbf{S}^{in}(z)$ for any $z \in \{0, 1\}^r$. Thus \mathbf{S}^{in} would not be collision-resistant, and in particular not collapsing.

We state the result formally:

Lemma 9 (Absorbing phase is collapsing). *Assume that \mathbf{f}^{right} is collapsing, and that \mathbf{f}^{right} is zero-preimage-resistant. Then \mathbf{S}^{in} is collapsing.*

Note that this lemma does not explicitly state anything about the size of r and c . But of course, \mathbf{f}^{right} can only be collapsing and zero-preimage-resistant if the capacity c is superlogarithmic.

We only give a detailed proof sketch for Lemma 9. The full proof is given in the full version [10].

Proof sketch. Consider a quantum-polynomial-time adversary (A, B) where A outputs a hash h , and a superposition of messages m on the register M , and B expects M back and outputs a guess b . We need to show that the two games in Definition 1 (see also Fig. 1) are indistinguishable, i.e., the probability of $b = 1$ is approximately the same in both games. Since the domain of \mathbf{S}^{in} is $(\{0, 1\}^r)^+$, we can assume that (A, B) is valid on $(\{0, 1\}^r)^+$, i.e., M contains a superposition of messages $m \in (\{0, 1\}^r)^+$ with $\mathbf{S}^{in}(m) = h$.

To show that the two games are indistinguishable, we start with Game_2 from Definition 1 (Fig. 1(b)) and transform it step by step into Game_1 .

Game 1 $(M, h) \leftarrow A()$. $b \leftarrow B(M)$. (Same as Game_2 from Definition 1.)

Note that we keep the register S (the state of (A, B)) implicit in this proof sketch, to improve readability.

Now, in each successive game, we measure more and more information about the message m contained in M , until in the final game, we measure m completely

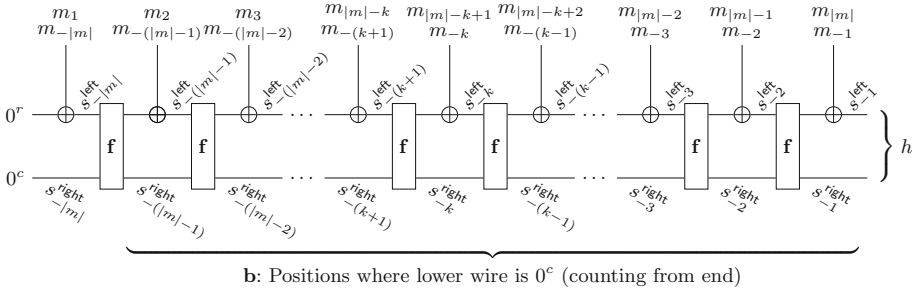


Fig. 3. Values occurring in the computation of $h = \mathbf{S}^{in}(m)$, using the notation from the proof sketch of Lemma 9.

(like in Game_1 from Definition 1). In order to refer to the different values derived from m that we measure, we will need to use a lot of notation to refer to various intermediate values occurring in the computation of $\mathbf{S}^{in}(m)$. To make it easier to follow the proof, all relevant notation has been depicted in Fig. 3, which shows an evaluation of $\mathbf{S}^{in}(m)$.

Since (A, B) is valid, we know that $\mathbf{S}^{in}(m) = h$ where h is the classical output of A . That is, $h = \mathbf{S}^{in}(m)$ has already been measured.⁸ In the computation of $\mathbf{S}^{in}(m)$, let s_{-2} refer to the state that goes into the last application of \mathbf{f} (see Fig. 3), and let m_{-1} refer to the last block of the input message m (i.e., $m_{-1} = m_{|m|}$). Then $h = \mathbf{f}(s_{-2}) \oplus (m_{-1} || 0^c)$ by definition of \mathbf{S}^{in} , and thus $h^{\text{right}} = \mathbf{f}^{\text{right}}(s_{-2})$. Now since by assumption, $\mathbf{f}^{\text{right}}$ is collapsing, we can reason: Since $\mathbf{f}^{\text{right}}$ is collapsing, and we have measured the output h^{right} of $\mathbf{f}^{\text{right}}(s_{-2})$, it follows that we can additionally measure s_{-2} , and the adversary B will not be able to notice the difference. That is, we get the following game with only negligibly different $\Pr[b = 1]$:

Game 2_{attempt} $(M, h) \leftarrow A()$. *Measure* s_{-2} . $b \leftarrow B(M)$.

This would indeed work, if we knew that $|m| \geq 2$. However, it could be that $|m| = 1$. In this case, we have $s_{-2} = \perp$ (i.e., s_{-2} does not occur in the computation). Worse, if M contains a superposition of messages m , some of length $|m| = 1$, others of length $|m| \geq 2$, then measuring s_{-2} will reveal whether $|m| \geq 2$ or $|m| = 1$. We cannot guarantee that this measurement will not change the quantum state of M in a noticeable way. Then $\Pr[b = 1 : \text{Game 1}] \not\approx \Pr[b = 1 : \text{Game 2}_{\text{attempt}}]$. The collapsing property of $\mathbf{f}^{\text{right}}$ does not help here, because to apply that property, we need to know that h^{right} is indeed the output of $\mathbf{f}^{\text{right}}$ (which is not the case when $|m| = 1$).

⁸ In this proof sketch, when we use the expression “measure a ” where a is some expression depending on the message m (e.g., a could be $\mathbf{S}^{in}(m)$), then we mean that we measure the register M , but not with a complete measurement, but with a measurement that gives outcome a (e.g., $\mathbf{S}^{in}(m)$) when M contains $|m\rangle$. Formally, that measurement would consist of the projectors P_i defined by $P_i := \sum_{m \text{ s.t. } a=i} |m\rangle\langle m|$. E.g., if we “measure $\mathbf{S}^{in}(m)$ ”, the projectors are $P_i := \sum_{m \text{ s.t. } \mathbf{S}^{in}(m)=i} |m\rangle\langle m|$.

A similar problem also occurs in later games: Let s_{-k} denote the k -th state from the end, with h being s_{-1} , the input to the last \mathbf{f} being s_{-2} , the input to the previous \mathbf{f} being s_{-3} , etc., see Fig. 3. (We count backwards because this will make notation easier, since our games will start measuring states from the end.) When we have measured some right state s_{-k}^{right} , we want to argue that we can measure the previous state $s_{-(k+1)}$ because $s_{-k}^{\text{right}} = \mathbf{f}^{\text{right}}(s_{-(k+1)})$. Again, this will not be possible because we do not know whether s_{-k} is not already the first state of the computation of $\mathbf{S}^{\text{in}}(m)$. (That is, we do not know whether $|m| = k$.)

To get around this problem, we need a mechanism to decide whether a state s_{-k} is the initial state, i.e., whether s_{-k} is $s_{-|m|}$. How do we do that? By construction of \mathbf{S}^{in} , the initial state $s_{-|m|}$ satisfies $s_{-|m|}^{\text{right}} = 0^c$. Thus, we might try to decide whether s_{-k} is the initial state by checking whether $s_{-k}^{\text{right}} = 0^c$. For example, if we want to measure s_{-2} , we do so only when $h^{\text{right}} = s_{-1}^{\text{right}} \neq 0^c$. This approach is basically sound, but what happens when a state in the middle has $s_{-k}^{\text{right}} = 0^c$? We would be misled, and the proof would break down.

To avoid this problem, we will first measure at which positions this bad case happens. Let \mathbf{b} be the set of all indices $k < |m|$ such that $s_{-k}^{\text{right}} = 0^c$. (That is, the indices of all states in which we observe a 0^c in the right part, but which are not the initial state.) Once we know the set \mathbf{b} , then we can decide whether s_{-k} is the initial state or not. Namely, s_{-k} is the initial state (i.e., $k = |m|$) iff $s_{-k}^{\text{right}} = 0^c$ and $k \notin \mathbf{b}$.

So the first step in our sequence of games is to measure the set \mathbf{b} :

Game 2 $(M, h) \leftarrow A()$. *Measure* \mathbf{b} . $b \leftarrow B(M)$.

We assumed that $\mathbf{f}^{\text{right}}$ is zero-preimage-resistant. This implies that with overwhelming probability, $s_{-k}^{\text{right}} = \mathbf{f}^{\text{right}}(s_{-(k+1)}) \neq 0^c$ for all $k < |m|$. Thus $\mathbf{b} = \emptyset$ with overwhelming probability. Therefore measuring \mathbf{b} has only negligible effect on the quantum state. Thus

$$\Pr[\text{Game 1}] \approx \Pr[\text{Game 2}].$$

(We use the shorthand $\Pr[\text{Game 1}]$ for $\Pr[b = 1 : \text{Game 2}]$. And \approx denotes a negligible difference.) Now we can proceed with measuring more and more states from the computation of $\mathbf{S}^{\text{in}}(m)$. First, we measure s_{-1} :

Game 4₁ $(M, h) \leftarrow A()$. *Measure* \mathbf{b} *and* s_{-1} . $b \leftarrow B(M)$.

(Note: The numbering of games in this proof sketch has gaps so that the game numbers here match the game numbers in the full version [10].) Since $s_{-1} = h$ by definition, and since h is already measured by A , the additional measurement does not change the quantum state, and we have:

$$\Pr[\text{Game 2}] = \Pr[\text{Game 4}_1].$$

Now we add a measurement whether s_{-2} is defined:

Game 5₁ $(M, h) \leftarrow A()$. Measure \mathbf{b} and s_{-1} and whether $s_{-2} = \perp$.⁹ $b \leftarrow B(M)$.

Given \mathbf{b} and s_{-1} we can already tell whether $s_{-2} = \perp$. Namely, $s_{-2} = \perp$ iff $s_{-1}^{\text{right}} = 0^c$ and $1 \notin \mathbf{b}$. Thus measuring whether $s_{-2} = \perp$ has no effect on the quantum state, and we get

$$\Pr[\text{Game 4}_1] = \Pr[\text{Game 5}_1].$$

Now, finally, we can do what we already intended to do in Game 2_{attempt}: We measure s_{-2} , and use the collapsing property of $\mathbf{f}^{\text{right}}$ to show that this measurement does not noticeably disturb the quantum state:

Game 4₂ $(M, h) \leftarrow A()$. Measure \mathbf{b} , and s_{-1} , and whether $s_{-2} = \perp$, and s_{-2} . $b \leftarrow B(M)$.

In case that we measured that $s_{-2} = \perp$, measuring s_{-2} in Game 4₂ has no effect on the quantum state (since we know that the outcome will be \perp). And in case that we measured that $s_{-2} \neq \perp$, we know that $s_{-1}^{\text{right}} = \mathbf{f}^{\text{right}}(s_{-2})$ (as already discussed above), and thus measuring s_{-2} can be noticed with at most noticeable probability by a quantum-polynomial-time adversary. Thus

$$\Pr[\text{Game 5}_1] \approx \Pr[\text{Game 4}_2].$$

And then we continue by adding a measurement whether $s_{-3} \neq \perp$:

Game 5₂ $(M, h) \leftarrow A()$. Measure \mathbf{b} , and s_{-1} , and whether $s_{-2} = \perp$, and s_{-2} , and whether $s_{-3} = \perp$. $b \leftarrow B(M)$.

Since $s_{-3} = \perp$ iff $s_{-2} = \perp$ or $s_{-2}^{\text{right}} = 0^c$ and $2 \notin \mathbf{b}$, measuring whether $s_{-3} = \perp$ holds has no effect on the quantum state. Thus we get

$$\Pr[\text{Game 4}_2] = \Pr[\text{Game 5}_2].$$

And then we measure s_{-3} :

Game 4₃ $(M, h) \leftarrow A()$. Measure \mathbf{b} , and s_{-1} , and whether $s_{-2} = \perp$, and s_{-2} , and whether $s_{-3} = \perp$, and s_{-3} . $b \leftarrow B(M)$.

Using that $\mathbf{f}^{\text{right}}$ is collapsing, we get

$$\Pr[\text{Game 5}_2] \approx \Pr[\text{Game 4}_3].$$

We continue in this way, alternatively adding a measurement whether the next state $s_{-k} = \perp$, and then adding a measurement of s_{-k} , each time using the collapsing property of $\mathbf{f}^{\text{right}}$. After ℓ such steps, where ℓ is a polynomial upper bound on the length of m , we get the following game:

Game 4_{\ell} $(M, h) \leftarrow A()$. Measure \mathbf{b} , measure whether $s_{-1}, \dots, s_{-\ell} = \perp$, measure $s_{-1}, \dots, s_{-\ell}$. $b \leftarrow B(M)$.

⁹ Measuring “whether $s_{-2} = \perp$ ” means a measurement on M defined by projectors P and $1 - P$ where $P := \sum_{m \text{ s.t. } s_{-2} = \perp} |m\rangle\langle m|$.

Since in each of the steps, we accrue only a negligible distinguishing probability between consecutive games, we get:

$$\Pr[\text{Game } 4_1] \approx \Pr[\text{Game } 4_\ell].$$

(Details are given in the full version [10].)

In Game 4_ℓ , we measure $s_{-1}, \dots, s_{-\ell}$. From these, we can compute $|m|$ (since $s_{-k} = \perp$ for $k > |m|$). Furthermore, each message block m_{-k} (the k -th message block from the end) can be computed as follows: We have $s_{-k} = \mathbf{f}(s_{-(k+1)}) \oplus (m_{-k} \| 0^c)$, and thus $m_{-k} = s_{-k}^{\text{left}} \oplus \mathbf{f}(s_{-(k+1)})^{\text{left}}$. Except for $m_{-|m|}$, which can be computed as $m_{-|m|} = s_{-|m|}^{\text{left}}$. (Cf. Fig. 3) Finally, we can compute m as $m = m_{-|m|} \| \dots \| m_{-1}$.

Since we can compute m from the measurements performed in Game 4_ℓ , it follows that those measurements are equivalent (in their effect on the quantum state) to a measurement of m . Thus

$$\Pr[\text{Game } 4_\ell] = \Pr[\text{Game } 6]$$

for the following final game:

Game 6 $(M, h) \leftarrow A()$. *Measure* m . $b \leftarrow B(M)$.

Altogether, we have shown

$$\Pr[\text{Game } 1] \approx \Pr[\text{Game } 6].$$

And Game 1 and Game 6 are identical to the games Game_2 and Game_1 from Definition 1, respectively. Since (A, B) was an arbitrary quantum-polynomial-time adversary that is valid for \mathbf{S}^{in} , it follows by Definition 1 that \mathbf{S}^{in} is collapsing. \square

Next, we show that the squeezing phase is collapsing. Let $\mathbf{f}^{\text{left}/n}$ be defined for $n > 0$, as the first $\min(n, r)$ bits of the output of \mathbf{f} (in particular, $\mathbf{f}^{\text{left}/n} = \mathbf{f}^{\text{left}}$ for $n \geq r$). Then the collapsing property of the squeezing phase is a relatively trivial consequence of the fact that $\mathbf{f}^{\text{left}/n}$ is collapsing.

Lemma 10 (Squeezing phase is collapsing). *Let $n > 0$ be the output length and assume that $\mathbf{f}^{\text{left}/n}$ is collapsing. Then \mathbf{S}^{out} is collapsing.*

A concrete security variant of this lemma is given in the full version [10].

Proof. Let $G_\eta(x)$ return the first $\eta = \min(r, n)$ bits of x . Then $G_\eta(\mathbf{S}^{\text{out}}(s)) = \mathbf{f}^{\text{left}/n}(s)$. Thus the lemma follows directly from Lemma 4. \square

And finally we get that the sponge construction as a whole is collapsing. This is a simple corollary from the fact that both the absorbing and the squeezing phase are collapsing.

Theorem 11 (Sponge construction is collapsing). *Let $n > 0$ be the output length and assume that $\mathbf{f}^{\text{left}/n}$ and $\mathbf{f}^{\text{right}}$ are collapsing, and that $\mathbf{f}^{\text{right}}$ is zero-preimage-resistant. Assume that pad is injective. Then \mathbf{S} is collapsing.*

A concrete security variant of this theorem is given in the full version [10].

Proof. By Lemma 9, \mathbf{S}^{in} is collapsing, and by Lemma 10, \mathbf{S}^{out} is collapsing. Then by Lemma 5, $\mathbf{S}^{out} \circ \mathbf{S}^{in}$ is collapsing. Since pad is injective, by Lemma 2, pad is collapsing. Thus by Lemma 5, $\mathbf{S} = (\mathbf{S}^{out} \circ \mathbf{S}^{in}) \circ pad$ is collapsing. \square

5.1 Using Random Oracles or Random Permutations

In this section, we show that \mathbf{S} is collapsing, when \mathcal{O} is a random function or random permutation and $\mathbf{f}^{\mathcal{O}}(x) := \mathcal{O}(x)$. The collapsing of $\mathbf{f}^{\text{right}}$, $\mathbf{f}^{\text{left}/n}$ follows from [18], and the zero-preimage-resistance of $\mathbf{f}^{\text{right}}$ follows from the optimality of Grover's algorithm. The computation of the precise advantage is given in the full version [10].

Theorem 12. *If $\mathcal{O} : \{0, 1\}^{r+c} \rightarrow \{0, 1\}^{r+c}$ is a random function, and $\mathbf{f}^{\mathcal{O}}(x) := \mathcal{O}(x)$, and r, c and output length n are superlogarithmic, then \mathbf{S} is collapsing.*

Proof. In the full version, we show that the preconditions of this Lemma follow from [18]. It then follows immediate from Theorem 11. \square

And since random functions and random permutations are known to be indistinguishable, we readily derive the security of the sponge construction also for block functions that are random permutations.

Theorem 13. *If $\mathcal{O} : \{0, 1\}^{r+c} \rightarrow \{0, 1\}^{r+c}$ is a random permutation, and $\mathbf{f}^{\mathcal{O}}(x) := \mathcal{O}(x)$, and r, c and output length n are superlogarithmic, then \mathbf{S} is collapsing.*

A concrete security variant (with security bounds in terms of the number of oracle-queries) is given in the full version [10].

Proof. Zhandry [20] shows that no adversary making a polynomial number of queries can distinguish a random permutation from a random function with more than negligible probability (assuming that the output length is superlogarithmic). Thus the advantage of the adversary attacking the collapsing property of \mathbf{S} when \mathcal{O} is a random permutation can only be negligibly higher than the advantage of the same adversary attacking the collapsing property of \mathbf{S} when \mathcal{O} is a random function. The latter advantage is negligible by Theorem 12, thus the former advantage is negligible, too. Hence \mathbf{S} is collapsing when \mathcal{O} is a random permutation. \square

6 Quantum Attack

In the following we present a quantum collision-finding attack against the Sponge construction. The attack is based on a quantum collision-finding algorithm for any function (Theorem 15 below) that assumes access to a random oracle (RO).

The general working of our attack is to select a *suitable* function g , run a collision finding algorithm by Ambainis [1] to obtain a collision for g , and finally turn this collision into a collision for the target Sponge. The *suitable* function in this context refers to the function giving the optimal result. First, we make a case distinction whether the length of the required collision n is smaller or bigger than the capacity c of the sponge. In case $n < c$, we simply search for an output collision in \mathbf{S} . In the other case $n \geq c$, it is more efficient to search for a right-collision, as these are collisions in a function with c bits of output and can be extended to arbitrary-length output collisions. Second, the function has to be selected or rather constructed in a way that allows for efficient iteration, in case a first run of the core algorithm does not succeed. Our attack makes heavy use of the following quantum algorithm by Ambainis [1].

Theorem 14 ([1] *Theorem 3*). *Let $\mathbf{g} : X \rightarrow Y$ be a function that has at least one collision. The size of the set X is M . Then there exists a constant k_{Amb} and a quantum algorithm AMB making $k_{\text{Amb}} \cdot M^{2/3}$ quantum queries to \mathbf{g} that finds a collision with probability at least $15/16$.*

We note that [1] also gives guarantees on the actual quantum running time and memory requirements of the quantum collision-finding algorithm. Concretely, there exist (small) constants $k'_{\text{Amb}}, k''_{\text{Amb}}$ such that the running time and quantum memory is at most $k'_{\text{Amb}} \cdot M^{2/3} \cdot \log^{k'_{\text{Amb}}}(M + |Y|)$. Therefore, all our results of this section which are stated in terms of query complexity also yield guarantees on the running time and memory, incurring the same blowup by a poly-logarithmic factor in the number of queries.

6.1 Quantum Collision Finding with Random Oracle

We start by showing how to use Ambainis' algorithm to generically find a collision in any function as long as we have access to a random oracle.

Theorem 15. *For finite sets A, B with $|B| \geq 3$ and $|A| \geq 40|B|$, and any function $h : A \rightarrow B$, there exists a quantum algorithm which requires access to a random oracle $\mathcal{H} : T \rightarrow A$ and outputs a collision of h with probability at least $1/8$ after at most $k_{\text{Amb}} \cdot |B|^{1/3}$ queries to h and at most $2k_{\text{Amb}} \cdot |B|^{1/3} + 2$ queries to \mathcal{H} where k_{Amb} is the constant from Theorem 14.*

As noted after Theorem 14, there exist constants $k'_{\text{Amb}}, k''_{\text{Amb}}$ such that the running time and quantum memory of the collision-finding algorithm is at most $k'_{\text{Amb}} \cdot |B|^{1/3} \cdot \log^{k''_{\text{Amb}}}(|B|)$.

Proof. Let $T := \{1, 2, \dots, \lceil \sqrt{|B|} \rceil + 1\}$ be a finite set of $\lceil \sqrt{|B|} \rceil + 1$ elements. In the description of our generic collision-finding algorithm COLL-RO below, we use the random oracle (RO) $\mathcal{H} : T \rightarrow A$. When repeating the algorithm in order to improve the success probability, we assume that a “fresh” random oracle is used in every run, which can be achieved using standard techniques such as prepending an iteration-counter to the inputs.

Algorithm 1. Algorithm COLL-RO

Input: $h : A \rightarrow B$ and access to random oracle $\mathcal{H} : T \rightarrow A$
Output: $m \neq \hat{m}$ such that $h(m) = h(\hat{m})$ or “fail”

- 1: Set $\mathbf{g} := h \circ \mathcal{H}$, $X := T$ with size $M = \lceil \sqrt{|B|} \rceil + 1$, $Y := B$
 - 2: Run AMB from Theorem 14 on \mathbf{g} , making $k_{\text{Amb}} \cdot M^{2/3}$ queries to \mathbf{g} .
 - 3: If it outputs (t, \hat{t})
 - 4: Set $(m, \hat{m}) := (\mathcal{H}(t), \mathcal{H}(\hat{t}))$
 - 5: If $m \neq \hat{m}$, output (m, \hat{m})
 - 6: Output “fail”
-

If Ambainis’ algorithm succeeds in outputting a collision and \mathcal{H} does not have any collisions, then we obtain a collision of h . Hence,

$$\begin{aligned}
 & \Pr[\text{COLL-RO outputs } m \neq \hat{m} \text{ such that } h(m) = h(\hat{m})] \\
 & \geq \Pr[\text{AMB outputs a collision} \wedge \mathcal{H} \text{ does not have collisions}] \\
 & \geq \Pr[\text{AMB outputs a collision}] - \Pr[\mathcal{H} \text{ has a collision}]. \tag{1}
 \end{aligned}$$

We can lower bound the first probability as follows:

$$\begin{aligned}
 \Pr[\text{AMB outputs a collision}] &= \Pr[\mathbf{g} \text{ has a collision} \wedge \text{AMB outputs a collision}] \\
 &= \Pr[\mathbf{g} \text{ has a collision}] - \Pr[\mathbf{g} \text{ has a collision} \wedge \text{AMB does not output a collision}] \\
 &\geq \Pr[\mathbf{g} \text{ has a collision}] - \frac{1}{16}.
 \end{aligned}$$

Note that \mathbf{g} maps M messages to B . If these outputs were distributed independently and uniformly, we could lower bound the collision probability with a birthday bound. In our case, these outputs are not necessarily uniformly (due to h) but still independently (due to \mathcal{H}) distributed. It is proven in [15] that in this case, the same lower bound on the collision probability remains true. Therefore, it follows (e.g. from [14, Lemma A.16]) that

$$\Pr[\mathbf{g} \text{ has a collision}] \geq \frac{M(M-1)}{4 \cdot |B|} \geq \frac{(M-1)^2}{4 \cdot |B|} \geq \frac{\left(\lceil \sqrt{|B|} \rceil\right)^2}{4 \cdot |B|} \geq \frac{1}{4}. \tag{2}$$

In order to upper bound the second term of (1), observe that \mathcal{H} maps M messages to independent and uniform elements in A . From a union bound (see, e.g. [14, Lemma A.15], noting that $M \leq \sqrt{2|A|}$), we get that

$$\Pr[\mathcal{H} \text{ has a collision}] \leq \frac{M^2}{2|A|} \leq \frac{(\sqrt{|B|} + 2)^2}{2|A|} \leq \frac{|B| + 4\sqrt{|B|} + 4}{2|A|} \leq \frac{5|B|}{2|A|} \leq \frac{1}{16}. \tag{3}$$

The second-to-last inequality holds because $4\sqrt{|B|} + 4 \leq 4|B|$ for $|B| \geq 3$, and the last inequality is due to our assumption $40|B| \leq |A|$. Combining (2) and

(3), COLL-RO outputs a collision with probability at least $\frac{1}{4} - \frac{1}{16} - \frac{1}{16} = \frac{1}{8}$. The quantum circuit for $\mathbf{g} := h \circ \mathcal{H}$ makes one query to h and two queries to \mathcal{H} . Therefore, the total number of queries to h is at most $k_{\text{Amb}} \cdot M^{2/3} = k_{\text{Amb}} \cdot |B|^{1/3}$ and the number of queries to \mathcal{H} is at most $2k_{\text{Amb}} \cdot |B|^{1/3} + 2$. \square

Theorem 16. *Let $\mathbf{S}_{c,r,\mathbf{f},\text{pad},n}(m)$ be a sponge construction with arbitrary block function \mathbf{f} . There exists a quantum algorithm COLL-RO making at most $q_{\mathbf{f}}$ quantum queries to \mathbf{f} and $q_{\mathcal{H}}$ quantum queries to a random oracle \mathcal{H} . COLL-RO outputs colliding messages $m \neq \hat{m}$ such that $\mathbf{S}_{c,r,\mathbf{f},\text{pad},n}(m) = \mathbf{S}_{c,r,\mathbf{f},\text{pad},n}(\hat{m})$ with probability at least $1/8$, where $q_{\mathbf{f}} := 2k_{\text{Amb}} \cdot \min\{\frac{c+6+2r}{r} 2^{c/3}, \frac{2n+6+3r}{r} 2^{n/3}\}$, and $q_{\mathcal{H}} := 2k_{\text{Amb}} \cdot \min\{2^{c/3}, 2^{n/3}\} + 2$, where k_{Amb} is the constant from Theorem 14 and pad is any padding function which appends at most $2r$ bits.*

Typical padding functions do not append more than $r + 1$ bits to the message, and are therefore covered by the theorem. Otherwise, the proof below can be easily modified to take longer paddings into account, resulting in increased factors in the expression of $q_{\mathbf{f}}$ above.

Proof. We make a case distinction whether the length n of the required collision is smaller or bigger than the capacity c of the sponge. In case $n < c$, it is more efficient to directly search for an output collision in \mathbf{S} . In the other case $n \geq c$, it is more efficient to search for collisions in the right internal state, as these are collisions in a function with c bits of output and can be extended to arbitrary-length output collisions.

Algorithm 2. Algorithm SPONGE-COLL-RO

Input: Sponge parameters n, c, r and access to RO \mathcal{H}

Output: $m \neq \hat{m}$ such that $\mathbf{S}_{c,r,\mathbf{f},\text{pad},n}(m) = \mathbf{S}_{c,r,\mathbf{f},\text{pad},n}(\hat{m})$ or “fail”

- 1: If $n < c$
 - 2: Set $h := \mathbf{S}$, domain $A := \{0, 1\}^{n+6}$ and range $B := \{0, 1\}^n$.
 - 3: If $n \geq c$
 - 4: Set $h := \mathbf{f}^{\text{right}} \circ \mathbf{S}^{\text{in}} \circ \text{pad}$, domain $A := \{0, 1\}^{c+6}$ and range $B := \{0, 1\}^c$
 - 5: Run COLL-RO from Theorem 15 on h , making $k_{\text{Amb}} \cdot |B|^{1/3}$ queries to h
 - 6: If it outputs (m, \hat{m})
 - 7: If $n < c$, output (m, \hat{m})
 - 8: If $n \geq c$
 - 9: Set $a := (\mathbf{f}^{\text{left}} \circ \mathbf{S}^{\text{in}} \circ \text{pad})(m) \oplus (\mathbf{f}^{\text{left}} \circ \mathbf{S}^{\text{in}} \circ \text{pad})(\hat{m})$
 - 10: Output $(\text{pad}(m) \| a, \text{pad}(\hat{m}) \| 0^r)$
 - 11: Output “fail”
-

Let us analyze the case $n < c$. According to Theorem 15, COLL-RO outputs a collision with probability at least $\frac{1}{8}$ using at most $k_{\text{Amb}} \cdot |B|^{1/3} = k_{\text{Amb}} \cdot 2^{n/3}$ quantum queries to h . A single evaluation of \mathbf{S} requires at most $2 \cdot \lceil \max\{|\text{pad}(m)| : m \in \{0, 1\}^{n+6}\} / r \rceil \leq 2 \cdot (n + 6 + 2r) / r$ queries to the block function \mathbf{f} in the absorbing phase and $2 \cdot \lceil n / r \rceil \leq 2(n + r) / r$ queries to \mathbf{f} in the squeezing phase.

Hence, one query to h requires at most $2 \cdot (2n+6+3r)/r$ queries to the block function \mathbf{f} . Therefore, a collision in \mathbf{S} can be found with at most $2k_{\text{Amb}} \cdot \frac{2n+6+3r}{r} 2^{n/3}$ queries to \mathbf{f} . In the other case $n < c$, the algorithm COLL-RO finds two messages $m \neq \hat{m}$ such that $(\mathbf{f}^{\text{right}} \circ \mathbf{S}^{in} \circ \text{pad})(m) = (\mathbf{f}^{\text{right}} \circ \mathbf{S}^{in} \circ \text{pad})(\hat{m})$ with probability at least $\frac{1}{8}$. Such a right-collision can then be extended to a full-state collision by appending to the padded colliding messages $\text{pad}(m)$ and $\text{pad}(\hat{m})$ one more suitably chosen message block resulting in $y := \text{pad}(m) \| a$ and $\hat{y} := \text{pad}(\hat{m}) \| 0^r$. As both $|y|$ and $|\hat{y}|$ are (possibly different) multiples of r , the same bits will be appended by pad according to our assumptions on the padding function. By the choice of a in Step 9, we have that $(\mathbf{f} \circ \mathbf{S}^{in} \circ \text{pad})(y) = (\mathbf{f} \circ \mathbf{S}^{in} \circ \text{pad})(\hat{y})$, i.e. the full states collide and therefore, all n output bits produced from this state will coincide. The algorithm makes $k_{\text{Amb}} \cdot |B|^{1/3} = k_{\text{Amb}} \cdot 2^{c/3}$ queries to $h := \mathbf{f}^{\text{right}} \circ \mathbf{S}^{in} \circ \text{pad}$. In this case, one query to h requires at most $2 \cdot (c+6+2r)/r$ queries to the block function \mathbf{f} . Therefore, a collision in $(\mathbf{f} \circ \mathbf{S}^{in} \circ \text{pad})$ can be found with at most $2k_{\text{Amb}} \cdot \frac{c+6+2r}{r} 2^{c/3}$ queries to \mathbf{f} , resulting in the claimed bound. \square

References

1. Ambainis, A.: Quantum walk algorithm for element distinctness. *SIAM J. Comput.* **37**(1), 210–239 (2007)
2. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: QUARK: a lightweight hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_1. ISBN 978-3-642-15030-2
3. Berger, T.P., D’Hayer, J., Marquet, K., Minier, M., Thomas, G.: The GLUON family: a lightweight hash function family based on FCSRs. In: Mitrokotsa, A., Vaudey, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 306–323. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31410-0_19. ISBN 978-3-642-31410-0
4. Bertoni, G., Daemen, J., Peeters, M., van Assche, G.: Sponge functions. In: *Ecrypt Hash Workshop*, May 2007. <http://sponge.noekeon.org/SpongeFunctions.pdf>
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indistinguishability of the sponge construction. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_11. ISBN 978-3-540-78966-6
6. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: the design space of lightweight cryptographic hashing. *IEEE Trans. Comput.* **62**(10), 2041–2053 (2013). <https://doi.org/10.1109/TC.2012.196>. ISSN 0018-9340
7. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3. ISBN 978-3-642-25384-3
8. Brassard, G., Hoyer, P., Tapp, A.: Quantum algorithm for the collision problem. arXiv preprint quant-ph/9705002 (1997)

9. Contini, S., Lenstra, A.K., Steinfeld, R.: VSH, an efficient and provable collision-resistant hash function. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 165–182. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_11. ISBN 978-3-540-34547-3
10. Czajkowski, J., Groot Bruinderink, L., Hülsing, A., Schaffner, C., Unruh, D.: Post-quantum security of the sponge construction. IACR ePrint 2017/711 (2017)
11. 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, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_13. ISBN 978-3-642-22792-9
12. Halevi, S., Micali, S.: Practical and provably-secure commitment schemes from collision-free hashing. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 201–215. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_16. ISBN 978-3-540-61512-5
13. Hülsing, A., Rijneveld, J., Song, F.: Mitigating multi-target attacks in hash-based signatures. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 387–416. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_15. ISBN 978-3-662-49384-7
14. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman & Hall/CRC Cryptography and Network Security Series, 2nd edn. Taylor & Francis, Milton Park (2014). ISBN 9781466570269
15. Knight, W., Bloom, D.M.: E2386. Am. Math. Mon. **80**(10), 1141–1142 (1973). <https://doi.org/10.2307/2318556>. <http://www.jstor.org/stable/2318556>. ISSN 00029890, 19300972
16. National Institute of Standards and Technology (NIST). Secure Hash Standard (SHS). FIPS PUBS 180-4. 2015. <https://doi.org/10.6028/NIST.FIPS.180-4>
17. NIST. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Draft FIPS 202 (2014). http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf
18. Unruh, D.: Computationally binding quantum commitments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 497–527. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_18. ISBN 978-3-662-49896-5
19. Unruh, D.: Collapse-binding quantum commitments without random oracles. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 166–195. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_6
20. Zhandry, M.: A note on the quantum collision and set equality problems. Quant. Inf. Comput. **15**(7&8), 557–567 (2015). <http://www.rintonpress.com/xxqic15/qic-15-78/0557-0567.pdf>



Putting Wings on SPHINCS

Stefan Kölbl^(✉)

DTU Compute, Technical University of Denmark,
Kongens Lyngby, Denmark
stek@dtu.dk

Abstract. SPHINCS is a recently proposed stateless hash-based signature scheme and promising candidate for a post-quantum secure digital signature scheme. In this work we provide a comparison of the performance when instantiating SPHINCS with different cryptographic hash functions on both recent Intel and AMD platforms found in personal computers and the ARMv8-A platform which is prevalent in mobile phones.

In particular, we provide a broad comparison of the performance of cryptographic hash functions utilizing the cryptographic extensions and vector instruction set extensions available on modern microprocessors. This comes with several new implementations optimized towards the specific use case of hash-based signature schemes.

Further, we instantiate SPHINCS with these primitives and provide benchmarks for the costs of generating keys, signing messages and verifying signatures with SPHINCS on Intel Haswell, Intel Skylake, AMD Ryzen, ARM Cortex A57 and Cortex A72.

Keywords: Post-quantum cryptography
Hash-based signature schemes · SPHINCS · Implementation · ARM

1 Introduction

Digital signature schemes are one of the fundamental cryptographic algorithms and are typically used to provide authenticity, integrity and non-repudiation for a message. They have found several applications in information security, e.g. certification of public keys, code signing or as an electronic signature. One of the major threats to the currently widely used digital signature schemes like DSA/ECDSA is that they are not secure if an attacker can build a large enough quantum computer. The security of these schemes relies on difficult number theoretic problems, which can be solved in polynomial time on a quantum computer [33].

There are various solutions for *post-quantum* secure digital signature schemes, namely lattice-based, multivariate-quadratic, code-based and hash-based signatures. One of the main advantage of hash-based signature schemes is that the security reduces to properties of the underlying cryptographic hash function. As every digital signature scheme requires a one-way function [32] these can be seen

as the minimal assumptions necessary to construct a secure signature scheme. All the other previously mentioned signature schemes require further assumptions by relying on the difficulty of *hard* problems for which the asymptotic difficulty might not always hold for the concrete instances used in a cryptographic systems and they require carefully choosing the parameters.

Hash-based digital signature schemes are therefore a very attractive choice. However most schemes, like XMSS [9] and LMS [14], are *stateful*, this means that one has to update the secret key with every signature. This may sound quite innocent, however it can be a severe difficulty in practice. For instance when sharing a private key on different computers one has to synchronize all of them or security can be void. For some applications this might be acceptable, however in general we desire to have a stateless signature scheme.

Goldreich proposed the first stateless hash-based signature scheme [18], however the parameters required for this construction to provide a sufficient security level and reasonable number of signatures per key pair resulting in a fairly large signature above 1 MB. SPHINCS [5] improves upon this construction in several aspects and first demonstrates that stateless schemes can be practical and provide a reasonable signature size (41 KB) while computing hundreds of signatures per second on a modern CPU.

The performance of SPHINCS directly relates to the underlying cryptographic hash function and therefore the performance of this function is critical, which will be the main focus of this work. The requirements for this function also differ from the classic use cases for cryptographic hash functions, as we do not require *collision resistance* and the inputs for most calls are rather short, typically 256 or 512 bits.

Contributions. The main goal of this work is to provide a comparison of performance when instantiating SPHINCS with different hash functions on modern high-end processors found in personal computers and mobile phones. In order to achieve this we provide several implementations, for modern Intel, AMD and ARM CPUs, optimized towards the requirements of SPHINCS. This includes implementations of SHA256, KECCAK, SIMPIRA, HARAKA and CHACHA optimized for hashing short inputs in parallel utilizing vector instructions and cryptographic extensions available on these microprocessors.

We further instantiate SPHINCS with these implementations and provide a broad comparison of the costs of generating key pairs, signing messages and verifying signature on Intel Haswell, Intel Skylake, AMD Ryzen, ARM Cortex A57 and A72. These are also the first optimized implementations for the ARMv8-A platform for SPHINCS and improve the understanding of the costs of stateless hash-based signature schemes. This performance results also indicate that SPHINCS is practical on the architecture used in a growing number of mobile phones.

Software. The implementations are put in the public domain and are available under <https://github.com/kste/sphincs>.

Related Work. So far there is only a limited amount of benchmarks for SPHINCS available. The original paper proposing SPHINCS [5] provides a reference implementation and an optimized implementation which utilizes the AVX2 vector extensions for speeding up the underlying CHACHA permutation. In [29] the authors propose a dedicated short-input hash function HAKA, which utilizes the AES instruction set to speed-up hash-based signature schemes and also provide some benchmarks for SPHINCS on the recent Intel platforms. The AES-based permutation design Simpira has recently also been proposed to instantiate SPHINCS [21] and its performance on Intel Skylake was evaluated. The first implementation on low-end platforms was provided in [25]. Here the authors demonstrate that SPHINCS can also be implemented on a 32-bit microcontroller based on the ARM Cortex M3 with very limited RAM available.

2 The SPHINCS Signature Scheme

In this section we give an overview of the SPHINCS digital signature scheme. Throughout the paper we will use the same parameters as suggested in [5], which will give a signature size of 41 KB, public-key size of 1056 bytes and a private-key size of 1088 bytes. These parameters target a security level of 128 bits against an adversary who has access to a large enough computer and allow up to 2^{50} signatures for a key pair. For more details we refer the reader to [5].

First, we will give a brief description of the main components used for SPHINCS and provide some insights on how much impact the performance of the underlying primitives has on the performance of SPHINCS. In particular, we are interested in two functions

$$\begin{aligned} \mathbf{F} &: \{0, 1\}^{256} \rightarrow \{0, 1\}^{256} \\ \mathbf{H} &: \{0, 1\}^{512} \rightarrow \{0, 1\}^{256}. \end{aligned} \tag{1}$$

which, as we will see later, are responsible for most of the computations in SPHINCS.

2.1 Hash Trees

At various points in the construction, SPHINCS uses a hash tree (also known as Merkle tree). A hash tree is a full binary tree of height h . We denote the i th node at level j of this tree as $N_{i,j}$, hence the root corresponds to $N_{0,h}$. Each node, which is not a leaf, gets labeled with the hash of its child nodes $N_{i,j} = \mathbf{H}(N_{2i,j-1} || N_{2i+1,j-1})$. Note that in order to drop the requirement for a collision resistant hash function [13], the inputs to \mathbf{H} are further masked in all hash trees used in SPHINCS.

An important term related with hash trees is the *authentication path*. The authentication path Auth_i serves as a proof that the node $N_{i,j}$ is part of the hash tree with root $N_{0,h}$. It contains the minimal number of nodes which are required to recompute the root of a hash tree given $N_{i,j}$. This newly computed root can then be compared with the previously committed one to verify that $N_{i,j}$ is indeed part of the original tree.

2.2 One-Time Signature: WOTS⁺

As a one-time signature SPHINCS uses WOTS⁺ [24], which has a parameter w allowing a trade-off between signature size and number of computations. Further, we derive the following parameters

$$l_1 = \left\lceil \frac{n}{\log w} \right\rceil, \quad l_2 = \left\lfloor \frac{\log(l_1(w-1))}{\log w} \right\rfloor + 1, \quad l = l_1 + l_2. \quad (2)$$

In the case of SPHINCS $w = 16$, thus $l = 67$. Additionally, we use \mathbf{F} to construct the chaining function

$$c^i(x) = \mathbf{F}(c^{i-1}(x) \oplus Q_i) \quad (3)$$

where Q_i is a round specific bitmask and $c^0(x) = x$.

Key Generation. The keys are derived from an initial secret key \mathcal{S} which is expanded using a pseudo-random generator (PRG) to obtain a sequence of secret keys $\text{sk} = (\text{sk}_1, \dots, \text{sk}_{67})$ for WOTS⁺. The public key pk is then computed by applying the chaining function on each part of the secret key

$$(\text{pk}_1, \dots, \text{pk}_{67}) = (c^{w-1}(\text{sk}_1), \dots, c^{w-1}(\text{sk}_{67})). \quad (4)$$

In order to reduce the size of this public key we build a hash tree on top of it to obtain pk . As l is usually not a power of two the L-tree [13] construction is used. This structure is similar to a binary tree, however if there is an odd number of nodes on a level the rightmost node is lifted up one level (see Fig. 1). The root of the resulting tree is then used as the public key pk .

Signing. A message m is signed by first computing the base w representation of the message $M = (M_1, \dots, M_{l_1})$. The next step is to compute a checksum $\sum_{i=1}^{l_1} (w-1-M_i)$ and also its base w representation $C = (C_1, \dots, C_{l_2})$. We concatenate these values and obtain $B = (B_1, \dots, B_l) = M || C$. The signature for M is then given by

$$\sigma = (\sigma_1, \dots, \sigma_l) = (c^{B_1}(\text{sk}_1), \dots, c^{B_l}(\text{sk}_l)). \quad (5)$$

Verification. The process of verifying a signature σ of a message m with the public key pk is done in a similar way. First, we have to recompute B and then compute

$$(\text{pk}'_1, \dots, \text{pk}'_l) = (c^{w-1-B_1}(\sigma_1), \dots, c^{w-1-B_l}(\sigma_l)) \quad (6)$$

Note that the correct bitmasks have to be used in each step of the chaining function to get the correct results. The final step is to recompute the root of the L-tree and check if $\text{pk}' = \text{pk}$.

2.3 Few-Time Signature: HORST

The second important component of SPHINCS is a few-time signature scheme. SPHINCS uses HORST, which is a variant of HORS [31] with an additional tree structure. HORST has two parameters t and k , which are $t = 2^{16}$ and $k = 32$ in the case of SPHINCS.

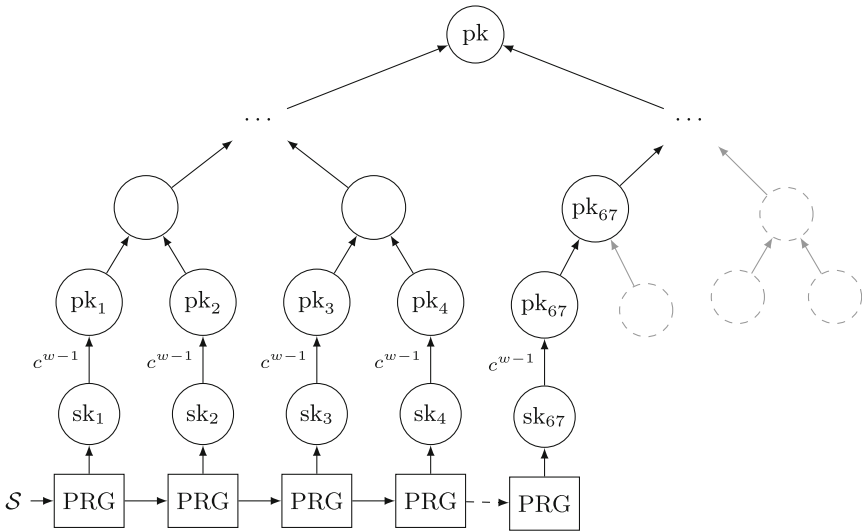


Fig. 1. WOTS⁺ key generation using an L-tree for computing the public key.

Key Generation. In order to generate the secret key we expand a secret S to obtain $sk = (sk_1, \dots, sk_t)$, similar to the WOTS⁺ key generation. The elements of this list are used to generate the leaves of a binary tree by computing $\mathbf{F}(sk_i)$. We then compute a hash tree on top of these leaves and the public key is the root node.

Signing. For signing, the message m is split into k pieces of length $\log t$ giving us $M = (M_1, \dots, M_k)$. Next, we interpret each M_i as an integer and compute the signature as $\sigma = (\sigma_1, \dots, \sigma_k, \sigma_{k+1})$. Each block $\sigma_i = (sk_{M_i}, \text{Auth}_{M_i})$ for all $i \leq k$. This corresponds to the M_i th element in the secret key and Auth_{M_i} are the elements required for computing the authentication path up to level 10 (see Fig. 2). Finally, σ_{k+1} contains all nodes at level 10 of the tree.

Verification. The verification process is very similar. First, the received parts of the secret key are hashed using \mathbf{F} . Together with the authentication paths this allows us to recompute the nodes at level 10 for each sk_i . These can then be verified with the values given in σ_{k+1} . Finally, the nodes in σ_{k+1} are used to recompute the root of the tree which has to be equal to pk .

2.4 Putting Everything Together

SPHINCS uses a nested tree structure consisting of 12 layers of trees of height 5 (see Fig. 3). Each tree is a binary tree where the leaves are the public key of a WOTS⁺ key pair. The top layer consists of a single tree and each key pair in the leaves is used to sign the root of another tree. Hence, on the second layer we

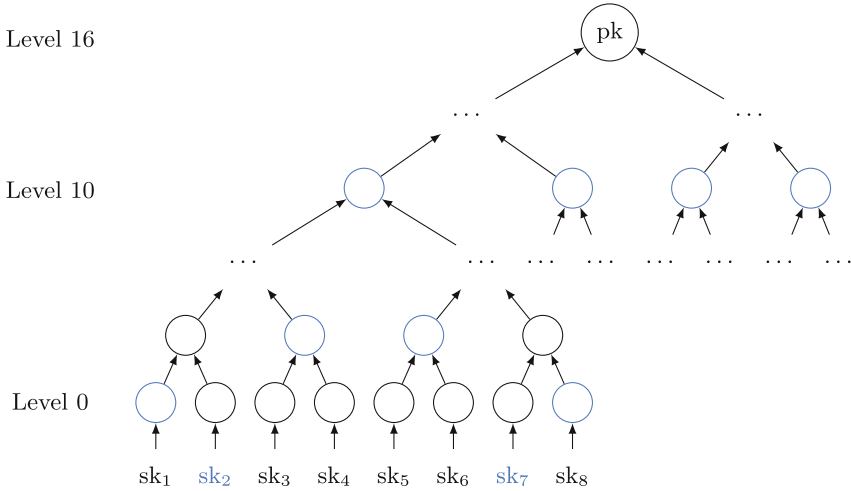


Fig. 2. Signing process in the HORST few-time signature scheme. In this case sk_2 and sk_7 are chosen by m and all the blue nodes are part of the authentication path and therefore part of the signature. (color figure online)

will have 32 trees. This process is repeated until we reach the bottom layer. On the bottom layer we use the final WOTS⁺ keys to sign a HORST public key, which is then used to sign the message.

Key Generation. For generating the keys in SPHINCS we choose two random 256-bit values $\mathcal{S}, \mathcal{S}'$. The first value is used during the key generation and the second one for signing. Furthermore, we need to generate all the bitmasks Q for WOTS⁺, HORST and the binary hash trees. For the public key pk we only need to compute the root of the tree at the top and therefore have to generate the 32 WOTS⁺ key pairs. The secret key is then $(\mathcal{S}, \mathcal{S}', Q)$ and the public key (pk, Q) .

Signing. The first step is to select a HORST key to sign the message. We use a pseudorandom function (which involves \mathcal{S}') to compute the index idx of the HORST key pair which we then use to sign a *randomized* digest R derived from m giving us the signature σ_{HORST} . Note that the HORST key pair is fully determined by this idx and the secret key \mathcal{S} .

The next step is to generate the WOTS⁺ key pair which signs the HORST public key used when computing σ_{HORST} . This again depends entirely on \mathcal{S} and the position in the tree and gives us the WOTS⁺ signature $\sigma_{w,1}$. The public key for this WOTS⁺ signature is part of another tree and needs to be authenticated again. We therefore compute the authentication path $Auth_{w,1}$ for $pk_{w,1}$.

This procedure of signing the root with a WOTS⁺ key pair and computing the authentication path is repeated until we reach the top layer. The full signature then consists of

$$\sigma = (idx, R, \sigma_{HORST}, \sigma_{w,1}, Auth_{w,1}, \dots, \sigma_{w,12}, Auth_{w,12}). \tag{7}$$

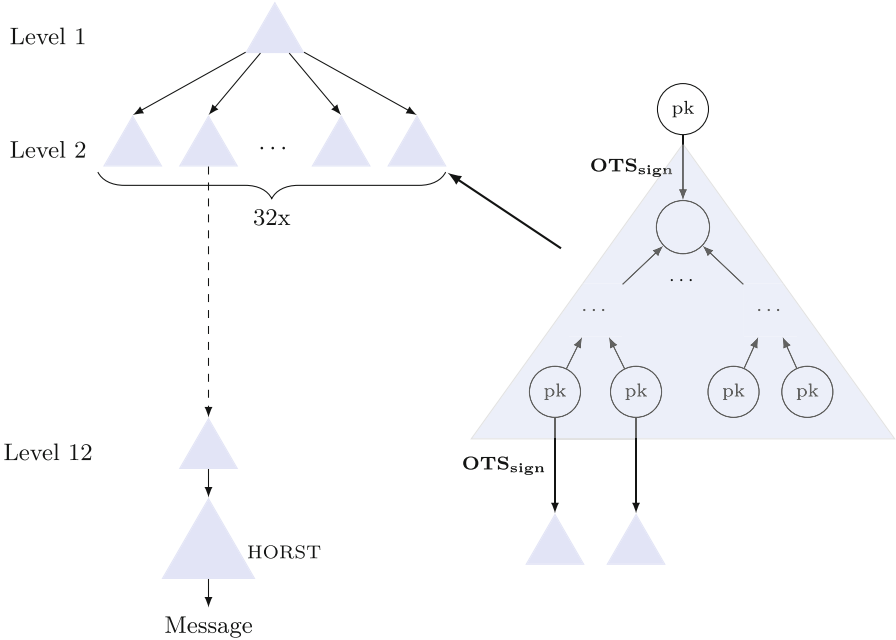


Fig. 3. Virtual tree structure used in SPHINCS.

Verification. The verification process consists of recomputing the randomized digest for the message and first verifying σ_{HORST} . If this is successful we continue with the verification of $\sigma_{w,1}$ and all further signature $\sigma_{w,i}$ until we reach the root of our tree. If all verifications succeed and the root of the top tree equals pk the signature is accepted.

3 How to Instantiate SPHINCS?

The performance of SPHINCS strongly correlates with the performance of two functions \mathbf{F} and \mathbf{H} which have the following security requirements

- **Preimage Resistance:** For a given output y it should be computationally infeasible to find an input x' such that $y = f(x')$.
- **Second-Preimage Resistance:** For a given x and $y = \mathcal{H}(x)$ it should be computationally infeasible to find $x' \neq x$ such that $f(x') = y$.
- **Undetectability:** It should be computationally infeasible for an adversary to predict the output.

For \mathbf{F} we require *preimage resistance*, *second-preimage resistance* and *undetectability*, while \mathbf{H} has to be *second-preimage resistant*. The best generic attacks against an ideal function with an output size of n bits require 2^n calls to the function respectively $2^{n/2}$ on a quantum computer using Grover's algorithm. In

the case of SPHINCS an attacker with access to a quantum computer should not be able to succeed in violating any of these properties with less than 2^{128} calls to the underlying function.

Contrary to a generic cryptographic hash function these requirements are very different. For instance we do not require those functions to be collision resistant, which in general is a much stronger requirement. Various cryptographic hash functions in the past have been broken in this setting like MD4 [35], MD5 [37] or SHA-1 [36] and while one can construct collisions in practice for all these functions, finding a preimage is still very costly, even for MD4 [22, 30]. The second difference is that these functions have a fixed input size. Most hash functions only reach their best performance for longer messages and several attacks are also only applicable for long messages.

Before, we discuss the different choices we first take a closer look at how many calls to these functions are required for *key generation*, *signing* and *verification* (also see Table 1). For generating the key in SPHINCS we need to do 32 WOTS⁺ key generations (and the corresponding L-tree) and construct the hash tree. In total this amounts to $32 \cdot (67 \cdot 15) = 32160$ computations of \mathbf{F} and $(32 \cdot 66) + 31 = 2143$ computations of \mathbf{H} .

Table 1. Costs in term of \mathbf{F} and \mathbf{H} for the operations in SPHINCS.

Operation	Calls to \mathbf{F}	Calls to \mathbf{H}
Key generation	32160	2143
Signing	451456	93406
Verification	≤ 12092	1235

For signing we need to compute one HORST signature and 12 trees which include the costs for one WOTS⁺ key generation each. Note that the WOTS⁺ signature can already be extracted while generating the WOTS⁺ key pairs. This means that one signature requires at least $65536 + (12 \cdot 32160) = 451456$ calls to \mathbf{F} and $65535 + 12 \cdot 2144 + 2143 = 93406$ calls to \mathbf{H} .

For verification we need one HORST verification and 12 WOTS⁺ verifications (including the L-tree) which corresponds to at most $12 \cdot (67 \cdot 15) + 32 = 12092$ calls to \mathbf{F} and $(12 \cdot (66 + 5)) + 383 = 1235$ calls to \mathbf{H} .

3.1 ChaCha

CHACHA is a family of stream ciphers [4]. In the original SPHINCS design both \mathbf{F} and \mathbf{H} are constructed from the 512-bit permutation π_{CHACHA} . If π_{CHACHA} represents 12 rounds of the CHACHA permutation then

$$\begin{aligned}\mathbf{F}(M_1) &= \text{Trunc}(\pi_{\text{CHACHA}}(M_1||C)) \\ \mathbf{H}(M_1||M_2) &= \text{Trunc}(\pi_{\text{CHACHA}}(\pi_{\text{CHACHA}}(M_1||C) \oplus (M_2||0^{256})))\end{aligned}$$

where M_1, M_2 are 256-bit messages and C is a 256-bit constant. `Trunc` is a function which truncates the output to 256 bits.

The best attack on the CHACHA stream cipher can recover a secret key for 7 rounds [2], however no concrete analysis exists in the construction used here. The building block used for the SHA-3 candidate BLAKE [3] shares a lot of similarities with the permutation used for CHACHA and it is likely that similar attack strategies can be applied. The best (second)-preimage attacks on BLAKE only cover 2.75 rounds and a (pseudo) preimage attack on 6.75 rounds of the compression function exists [16].

3.2 SHA256

SHA256 is one of the most widely used cryptographic hash functions. It was published in 2001 and designed by the NSA. The compression function processes blocks of 512-bit using the Davies-Meyer construction and can be directly used to build both \mathbf{F} and \mathbf{H} ¹. We denote these functions as SHA256- \mathbf{F} and SHA256- \mathbf{H} . The best preimage attacks on SHA256 reach 45 out of 64 rounds [28] and are only slightly faster than bruteforce. In [1], the costs of finding a preimage using Grover's quantum algorithm [19] for SHA-256 have been estimated at around 2^{166} basic operations.

3.3 Keccak

KECCAK is a family of cryptographic hash functions based on the Sponge construction and has been standardized as SHA-3 (FIPS PUB 202). It offers a range of permutations of size $b = 25 \cdot 2^l$ for $l = 0, \dots, 6$. For an output size of 256-bit the SHA-3 standard specifies to use KECCAK[$b = 1600, c = 512$]. This would allow us to instantiate \mathbf{F} and \mathbf{H} with a single call to the permutation, as we can process up to 1088 bits. However, this seems quite an inefficient use and it might be beneficial to use a smaller permutation. Recently, two versions of KECCAK with a reduced number of rounds have been proposed [8]. KANGAROOTWELVE for 128-bit security and MARSUPILAMIFOURTEEN for 256-bit security.

The *capacity* c in a sponge directly relates to the security level and in the classical setting a Sponge requires $c = 512$, to have 256-bit second-preimage resistance. However, it is not clear whether we need a capacity of 512 bits if we only require 2^{128} security against a quantum adversary.

In order to evaluate the potential of using KECCAK in SPHINCS we choose both a smaller permutation and reduce the number of rounds

$$\begin{aligned} \text{KECCAK-}\mathbf{F}(M) &= \text{Trunc}(\text{KECCAK}[b = 800, \text{rounds} = 12, c = 256](M)) \\ \text{KECCAK-}\mathbf{H}(M) &= \text{Trunc}(\text{KECCAK}[b = 800, \text{rounds} = 12, c = 256](M)). \end{aligned} \quad (8)$$

The best preimage attacks on KECCAK with an output size of 256-bit can cover 4 rounds of KECCAK [23], apart from a slight improvement over brute force

¹ To separate the domains of the two functions one could use a different IV or round constants.

with huge memory for 8 rounds [10]. The costs of applying Grover’s quantum algorithm to find a preimage for SHA3-256 have also been estimated at around 2^{166} in [1]. Overall, taking into account the restricted setting a reduced-round version of KECCAK seems reasonable for this use case.

3.4 Haraka

HARAKA is a short-input hash function, specifically designed for the use in hash-based signature schemes [29]. The construction uses an efficient 256-bit (resp. 512-bit) permutation based on the AES with a simple mode (see Fig. 4) to build the two functions **F** and **H**.

The best preimage attacks by the authors can find a preimage for 3.5 respectively 4 out of 5 rounds. For an earlier version of HARAKA-**H** there also exists an attack exploiting weak round constants which can find a preimage in 2^{192} evaluations [26], however this attack is not applicable to the current version.

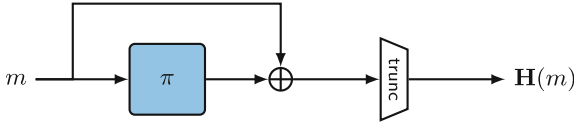


Fig. 4. Using a permutation π to construct a short-input hash function.

3.5 Simpira

SIMPIRA is a family of cryptographic permutations [20] that supports an input size of $b \cdot 128$. The design is based on generalized Feistel networks and uses the AES round function for updating the branches. The variants with $b = 2$ and $b = 4$ can be used in the same mode as HARAKA to construct

$$\begin{aligned} \text{SIMPIRA-F}(M) &= \text{SIMPIRA}[b = 2](M) \oplus M \\ \text{SIMPIRA-H}(M) &= \text{Trunc}(\text{SIMPIRA}[b = 4](M) \oplus M). \end{aligned} \quad (9)$$

The security claim for SIMPIRA is that no distinguisher with costs $< 2^{128}$ exists, but so far no concrete preimage attacks have been published.

4 Efficient Implementations for **F** and **H**

The target platforms for our implementations are on one hand the recent x86 CPUs by Intel (Haswell and Skylake), AMD (Ryzen) and on the other hand the ARMv8-A architecture, which has a large share in the mobile phone market. In order to understand how to efficiently implement our primitives on these platforms we give a quick overview of the most important features we utilize.

4.1 Instruction Pipeline

Modern CPUs have an instruction pipeline, which allows some form of parallelism on a single CPU core. This is realized by splitting up an instruction into different stages which can be executed in the same cycle. In order to assess the performance of an instructions we use two notions, the *latency* and the *inverse throughput*. Latency corresponds to the number of clock cycles we have to wait until we get the result of an instruction, while the inverse throughput is the number of clock cycles we have to wait until we can issue the same instruction again.

Utilizing the pipeline is an important performance consideration and can especially be useful for instructions with a high latency and low inverse throughput. This has previously been studied in various AES-based designs [20, 27, 29] to increase the performance of cryptographic operations. In the case of SPHINCS it is particularly easy to keep the pipeline filled up, as one has multiple independent inputs available for most operations. For instance, the WOTS⁺ chains can be computed in parallel and most levels of a hash tree allow a high degree of parallelism.

4.2 Vector Instructions

Another important feature of modern microprocessors are vector units which provide parallelism through *single instruction, multiple data* (SIMD) instructions. These instructions allow to apply the same operation to multiple values stored in a *vector register* and can significantly increase the throughput. For many cryptographic primitives the fastest implementations utilize SIMD instructions. While we often have to pack the data in a specific format, these costs are compensated by processing multiple messages/blocks in parallel. Especially in the case of hash-based signature where multiple independent inputs are almost constantly available it allows us to fully utilize this feature for a very efficient implementation.

On the current Intel and AMD platforms² the vector extensions is called AVX2, which features 16 registers of 256-bit. This will be further extended to AVX-512³, allowing to operate on 512-bit vectors which will likely speed-up all vector implementations through the higher degree of parallelism.

The ARMv8-A architecture offers the NEON instruction set, which allows to operate on 128-bit vectors. Future ARM platforms [34] will come with a scalable vector extension (SVE), supporting vectors up to a size of 2048 bits and hence allowing 16 times the parallelism compared to the current ARM processors.

² AVX2 is available since Intel Haswell, for older platforms the predecessor AVX can be used which supports 128-bit vectors.

³ AVX-512 can already be found in Xeon Phi (Knights Landing) and Skylake-X processors.

4.3 Crypto Extensions

An increasing number of platforms provide instructions carrying out cryptographic operations, which provide a significant speed-up for the supported primitives while also providing a constant running time and protection against cache-timing attacks. All recent Intel platforms provide instructions for the round function of the AES and a similar extensions is available on ARMv8-A. Additionally, the ARM crypto extensions support SHA-1 and SHA256. On the newest AMD platform Ryzen these instructions are also available and support for them is also planned for the next generation of Intel processors. An overview of these instructions and their performance characteristics is given in Table 3.

4.4 ChaCha-F and -H

The CHACHA permutation is very fast in software and benefits strongly from the SIMD features on modern CPUs, which is also one of the main motivations why the SPHINCS designers use it for instantiating SPHINCS. As the design is based on 32-bit words, AVX2 can be utilized to process up to 8 blocks in parallel. Similar, using ARM NEON we can process 4 blocks in parallel. On Intel platforms we use the original AVX2 implementation of CHACHA provided with SPHINCS in [6]. For ARM we use the implementation by Romain Dolbeau available in Supercop [6], as it is the fastest available using on the ARM Cortex A57, to construct ChaCha-F and ChaCha-H.

4.5 SHA256-F and -H

SHA256 is also based around operations on 32-bit words and therefore benefits in the same way as CHACHA from the use of SIMD instructions. For Intel Haswell and Skylake we implemented SHA256 using AVX2 processing 8 blocks in parallel.

We use eight registers, where each one contains one 32-bit word of the state S_i for all eight blocks (see Fig. 5). We assume that the incoming message blocks lie consecutively in memory and load them into 16 256-bit vectors. In order to have an efficient implementation of the message expansion we have to transpose the content of these vectors. This adds an overhead of 32 pack/unpack and 16 permute instructions. Note that this is not required for the state words, as we can simply use the transposed initial value.

By using this data representation the round function and message expansion can be implemented very efficiently and we require the same number of operations as none vectorized implementation. In order to get the correct output format we have to again transpose the state which adds another 16 pack/unpack and 8 permute instructions.

For AMD Ryzen and ARM we use the SHA256 crypto extensions as those implementations compare favorable in performance. The latency of these instructions is fairly high on both platforms and therefore we interleave four calls in parallel to make use of the instruction pipeline.

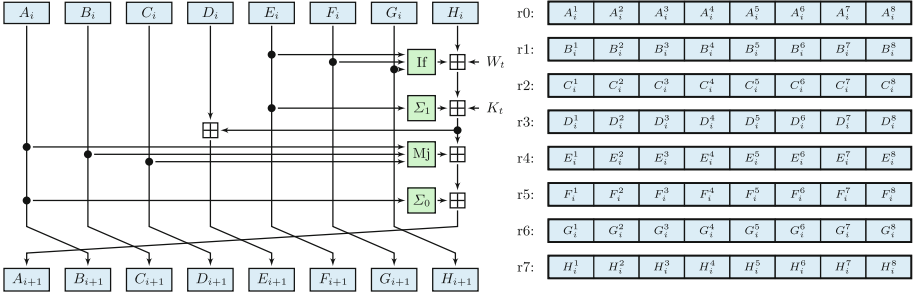


Fig. 5. On the left the SHA256 state update and on the right the mapping of the state for the eight blocks to the registers. A_i^j corresponds to the word A as input to round i for block j .

4.6 Keccak-F and -H

KANGAROOTWELVE already utilizes SIMD instructions and we base our construction of KECCAK-F and KECCAK-H on the available implementation [7] of KECCAK[$b = 1600, r = 12$] processing 4 blocks in parallel. The same strategy can be used to implement KECCAK[$b = 800, r = 12$] processing 8 blocks in parallel. Compared to SHA-3 as defined in FIPS PUB 202 we can gain a factor of ≈ 4 in speed as we can process double the number of blocks with half the number of rounds when using KECCAK[$b = 800, r = 12$].

For ARM we can use a similar approach, however only 2 (for KECCAK[$b = 1600, r = 12$]) resp. 4 blocks can be processed in parallel. For hashing a single input we use the ARMv8 implementation provided in the Keccak Code package [7] and for multiple inputs we implemented a version of KECCAK[$b = 800, r = 12$] processing four blocks in parallel using a strategy similar to the x86 implementation.

4.7 Haraka

For x86 we use the latest version of HARAKA available online⁴ and the only difference between the platforms is to find the optimal number of parallel calls. Depending on the platform it is better to interleave four or eight calls to HARAKA-F resp. HARAKA-H, which is related to the latency of the `aesenc` instruction (see Table 3). We therefore use eight calls in parallel on Haswell and four on Skylake/Ryzen.

One of the main difference between the AES instruction set on Intel and ARM is that on ARM one round of AES is split up into two instructions `aese` and `aesm`. It is very important that these two instructions are adjacent, as this significantly reduces the latency⁵. Another difference is that on Intel the key addition happens at the end of the round which aligns with the HARAKA

⁴ See <https://github.com/kste/haraka>.

⁵ see ARM Cortex A57 Software Optimization Guide, Page 35.

specification while the `aese` instruction on ARM adds the key at the beginning. One AES round on these platforms is therefore defined as

$$\begin{aligned} \text{aesenc} &= \text{AddKey} \circ \text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes} \\ \text{aesmc} \circ \text{aese} &= \text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes} \circ \text{AddKey}. \end{aligned} \tag{10}$$

For an efficient implementation we can use a different set of round constants to take this into account. HARAKA-256 uses the round constants RC_{2i} and RC_{2i+1} in the i th AES layer. In the ARM implementation we use an all zero constant for the first call and RC_0, RC_1 for the second layer. For the third AES layer we compute $RC'_2 || RC'_3 = \text{mix}_{256}^{-1}(RC_2, RC_3)$ (see Fig. 6). For the mixing operation used in HARAKA we can replace pack/unpack with the equivalent instruction on ARM `zip1` and `zip2`.

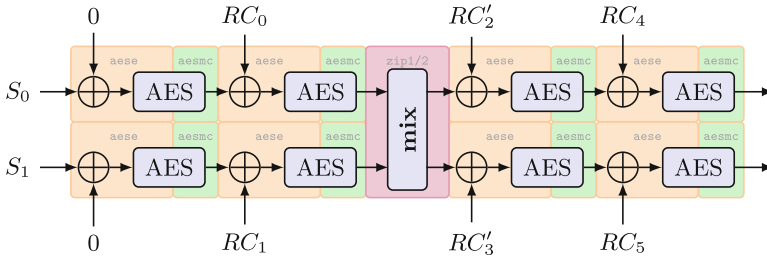


Fig. 6. Implementation of HARAKA-256 on ARM using the AES specific instructions. The order of `mix` and the addition of round constants are exchanged to facilitate the free XOR from the key addition of `aese`.

4.8 Simpira

SIMPIRA is another design which utilizes the AES round function, but in a Feistel network and therefore can be implemented with the AES instructions available on both x86 and ARM. The key addition is used to add a constant and to realize the XOR in the Feistel. On Intel we use the implementation provided by the SIMPIRA designers⁶ while for ARM we provide a new implementation.

Similar to the case of HARAKA it is important to have `aese` and `aesmc` aligned and interleave the calls to hide the latency. Also the different order of the key addition needs to be taken into account, which requires an additional XOR per round for $b = 2$ respectively two XORs for $b = 4$ to realize the Feistel networks used in SIMPIRA. In the x86 implementation these XORs are for free as the key addition happens at the end of `aesenc` which can be used to XOR with the other branches. Overall this adds a slight overhead compared on the ARM platforms, but still allows a very efficient implementation.

⁶ See <http://mouha.be/simpira/>.

5 Performance Results

We base our implementation of SPHINCS on the source code provided by the SPHINCS authors, which is also available in [6], and instantiate \mathbf{F} and \mathbf{H} with the previously discussed primitives to measure the number of cycles required to perform key generation, signing and verification.

The platforms we use for benchmarking include an Intel Haswell (i7-4770S with 3.1 GHz), an Intel Skylake (i7-6700 with 3.4 GHz), an AMD Ryzen (1700 with 3.7 GHz), ARM Cortex A57 (Samsung Galaxy S6 with 2.1 GHz) and an ARM Cortex A72 (Samsung Chromebook Plus with 2.0 GHz). All benchmarks are done on a single core and any frequency scaling technologies like Turbo Boost are deactivated. For measuring the cycle count we use the available performance counter on Intel/AMD and the wall-clock time on ARM. For compiling we use gcc version 6.3.0 with the flags `-O3 -mavx2 -march=native -mtune=native -fomit-frame-pointer` on Intel/AMD and for ARM we crosscompile with `-O3 -mcpu=A57+crypto -fomit-frame-pointer`.

As a first step we measured the performance of \mathbf{F} and \mathbf{H} for all our primitives on all platforms (see Figs. 7 and 8). We only highlight here the performance for processing multiple inputs in parallel, as in SPHINCS only a minority of the operations can not be parallelized. For single inputs the performance drops especially for the otherwise vectorized implementations of CHACHA, KECCAK and SHA256 (on Intel). In general the gap between the implementations utilizing crypto specific instructions and the vectorized implementations is much smaller on Intel than on ARM. Especially, KECCAK suffers from the smaller vector size and the higher latency and worse throughput of the vector instructions on ARM (see Table 3).

The performance numbers of these functions reflect directly in the costs for carrying out key generation, signing and verification in SPHINCS. In Table 2, we give an overview of the exact number of cycles required for each operation for the different instantiations of SPHINCS. Unsurprisingly, signing is the most

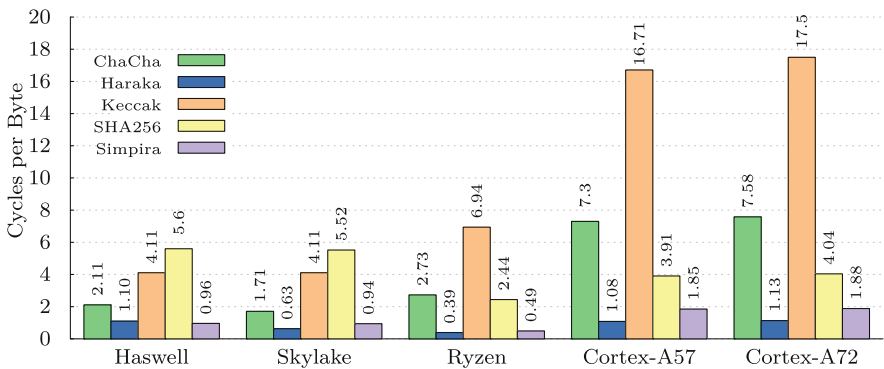


Fig. 7. Performance of \mathbf{F} on different platforms for processing multiple inputs in parallel. All numbers given are in cycles per byte.

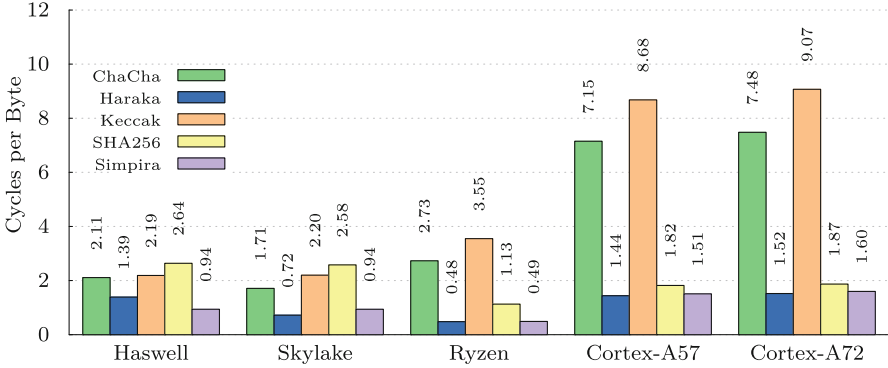


Fig. 8. Performance of **H** on different platforms for processing multiple inputs in parallel. All numbers given are in cycles per byte.

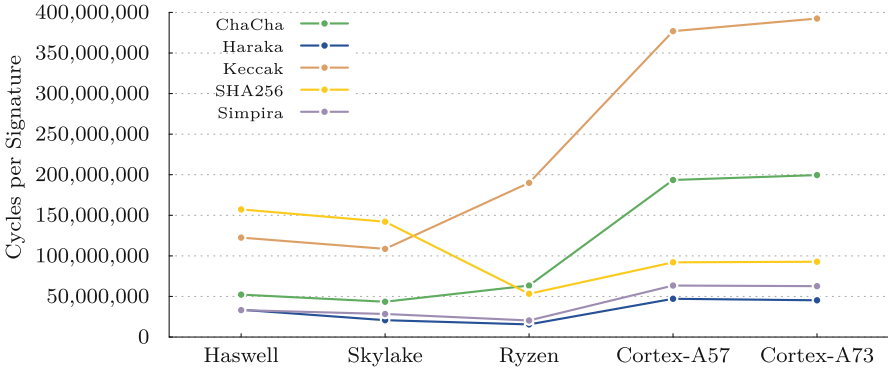


Fig. 9. Number of cycles for signing one message.

costly operation and allows the biggest gains for highly optimized designs like HARAKA and SIMPIRA. As we can see in Table 1, signing requires to call **F** five times more often than **H** and therefore the performance for **F** is of greater importance (Fig. 9).

On ARMv8-A the gap between the performance of the primitives without hardware support (CHACHA and KECCAK) and those with is much wider. SPHINCS-HARAKA is around eight times faster for signing than SPHINCS-KECCAK on the ARM Cortex A57, while the biggest gap on Skylake is only a factor of five. This again comes with no surprise, as the underlying functions exhibit a similar difference in performance on this platform. The performance of SPHINCS on mobile devices with the ARM Cortex A57 is very practical and on the Samsung Galaxy S6 used here which has four cores we can compute over hundred signatures per second for the SPHINCS instantiations which utilize hardware support.

Table 2. Benchmarks of SPHINCS on different platforms. All results are the median value of 100 measurements.

Architecture	Primitive	KeyGen	Sign	Verify
Intel Haswell	CHACHA	3.295.808	52.249.518	1.495.416
	HARAKA	2.027.136	33.640.796	592.036
	KECCAK	7.564.068	122.517.136	2.366.644
	SHA256	9.676.984	157.270.152	3.804.288
	SIMPIRA	2.108.364	33.210.104	595.524
Intel Skylake	CHACHA	2.839.018	43.495.454	1.291.980
	HARAKA	1.340.338	20.782.894	415.586
	KECCAK	6.589.798	108.629.952	2.152.066
	SHA256	8.724.516	142.063.840	2.812.466
	SIMPIRA	1.808.830	28.408.658	520.832
AMD Ryzen	CHACHA	3.648.660	63.427.980	1.587.120
	HARAKA	965.430	15.545.370	258.660
	KECCAK	11.354.460	189.986.970	3.739.140
	SHA256	3.267.180	53.332.380	1.090.650
	SIMPIRA	1.261.590	20.439.600	335.790
ARM Cortex A57	CHACHA	10.361.344	193.512.960	3.488.256
	HARAKA	2.246.656	47.100.928	717.824
	KECCAK	22.006.272	376.908.288	7.358.464
	SHA256	5.292.032	92.088.832	1.679.872
	SIMPIRA	3.362.304	63.489.536	1.108.992
ARM Cortex A72	CHACHA	10.940.928	199.582.208	3.666.944
	HARAKA	2.320.384	45.261.312	737.280
	KECCAK	22.963.712	392.445.952	7.640.064
	SHA256	5.359.616	92.767.744	1.717.760
	SIMPIRA	3.412.480	62.707.712	1.131.520

5.1 Comparison with Other Signature Schemes

To put the performance of SPHINCS into context with other recently proposed post-quantum digital signature schemes we provide a short overview with some of the candidates submitted to the NIST post-quantum competition. For most schemes there is only a limited amount of benchmarks available and those implementations are usually only optimized for x86. We therefore restrict this comparison to the platforms where optimized implementations exist.

Dilithium is a lattice-based signature scheme based on module lattices [15]. The set of parameters for which the authors claim 128-bit post-quantum security leads to a signature size of 2.7 kB. The authors also provide an optimized implementation utilizing AVX2 which on Haswell takes 251.590 cycles for key generation, signing 112.716.000 and verification 58.680.000.

Another candidate for lattice-based signature schemes is Falcon [17], which is based on the *short integer solution* problem over NTRU lattices. Choosing parameters which provide a similar security level as SPHINCS leads to a signature size of 1.2 kB. The authors also provide benchmarks on Skylake: Key generation takes 64.812.000, signing 1.074.219 and verification 186.472 cycles.

MQDSS [12] is a signature scheme based on the problem of solving multivariate quadratic equations. For the 128-bit post-quantum security level the signature size is comparable to SPHINCS at 41 kB. An optimized AVX2 implementation exists and on Haswell the scheme achieves a performance of 1.826.612 cycles for key generation, 8.510.616 for signing and 5.752.612 for verification.

A new digital signature scheme, based on non-interactive zero-knowledge proofs, named *Picnic* has been proposed [11]. The security also is based on the security of symmetric-key primitives similar to SPHINCS⁷. For the proposed parameters and instantiation Picnic has a signature size of 195 kB, however contrary to SPHINCS the size of the signature is also influenced by the choice of the symmetric-key primitive. The authors provide benchmarks on Haswell for Picnic instantiated with LowMC: Key generation takes 36.000, signing 112.716.000 and verification 58.680.000 cycles.

6 Conclusion

We presented a detailed discussion of how to instantiate SPHINCS, what the requirements are and how the performance relates to the underlying cryptographic hash function. Further, we provide an overview of promising candidates for instantiating SPHINCS and discuss their security and performance characteristics.

We provided benchmarks on Intel Haswell, Intel Skylake and ARM Cortex A57 for these primitives based on implementations optimized towards the requirements for hash-based signature schemes. Further, we provided a comparison of SPHINCS instantiated with those primitives.

Overall we can see that on current platforms the performance for primitives utilizing the crypto extensions is favorable compared to others and also the difference between Intel and ARMv8-A is smaller. However, all primitives relying on vectorized implementations get a significant slow down on ARMv8-A. Future platforms, with support for larger vectors, are in the pipeline and will very likely give a significant performance boost to hash-based signature schemes and will make those primitives more competitive.

Acknowledgments. We would like to thank Christoffer Brøndum for providing a first version of the ARM implementation of Haraka and Jacob Appelbaum for running the benchmarks on the Cortex A72.

This work was supported by the Commission of the European Communities through the Horizon 2020 program under project number 645622 (PQCRYPTO).

⁷ The main difference is that SPHINCS has a security proof in the standard model and Picnic in the quantum random-oracle model (QROM).

A Instructions

In Table 3 we give an overview of the performance characteristics^{8,9} of the instructions on the different platforms. Note that on the ARM Cortex A57/A73 a pair of `aese` and `aesmc` will have a latency of 3 and inverse throughput of 1.

Table 3. Comparison of the latency L and inverse throughput T of several instructions used in the implementations.

Instruction	Platform	L	T	Description
<code>vpxor, vpand, vpor</code>	Haswell	1	0.33	XOR/AND/OR of 256-bit vectors
	Skylake	1	0.33	
	Ryzen	1	0.5	
<code>veor, vand, vorr</code>	Cortex A57	3	2	XOR/AND/OR of 128-bit vectors
	Cortex A72	3	2	
<code>vpslld</code>	Haswell	1	1	Shift of words in 256-bit vectors
	Skylake	1	1	
	Ryzen	1	2	
<code>vshl</code>	Cortex A57	3	1	Shift of words in 128-bit vector
	Cortex A72	3	1	
<code>punpckhdq, punpckldq</code>	Haswell	1	1	Interleave upper/lower halves of two 128-bit vectors
	Skylake	1	1	
	Ryzen	1	0.5	
<code>zip1, zip2</code>	Cortex A57	3	2	
	Cortex A72	3	2	
<code>aesenc</code>	Haswell	7	1	SubBytes, ShiftRows, MixColumns, AddKey
	Skylake	4	1	
	Ryzen	4	0.5	
<code>aese, aesmc</code>	Cortex A57	3	1	AddKey, SubBytes, ShiftRows / MixColumns
	Cortex A72	3	1	
<code>SHA256RND52</code>	Ryzen	4	2	Two rounds of SHA256
<code>SHA256MSG1</code>	Ryzen	2	0.5	Helper for message expansion
<code>SHA256MSG2</code>	Ryzen	3	2	
<code>sha256h</code>	Cortex A57/A72	6	1	SHA256 state update
<code>sha256h2</code>	Cortex A57/A72	6	1	
<code>sha256su0</code>	Cortex A57/A72	3	1	SHA256 message expansion
<code>sha256su1</code>	Cortex A57/A72	6	1	

⁸ For Intel/AMD see: <https://software.intel.com/sites/landingpage/IntrinsicsGuide> and http://agner.org/optimize/instruction_tables.pdf.

⁹ For ARM see: http://infocenter.arm.com/help/topic/com.arm.doc.uan0015b/Cortex_A57_Software_Optimization_Guide_external.pdf.

References

1. Amy, M., Matteo, O.D., Gheorghiu, V., Mosca, M., Parent, A., Schanck, J.: Estimating the cost of generic quantum pre-image attacks on sha-2 and sha-3. Cryptology ePrint Archive, Report 2016/992 (2016). <http://eprint.iacr.org/2016/992>
2. Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of Latin dances: analysis of salsa, chacha, and rumba. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 470–488. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_30
3. Aumasson, J., Meier, W., Phan, R.C., Henzen, L.: The Hash Function BLAKE. Information Security and Cryptography. Springer, Heidelberg (2014). <https://doi.org/10.1007/978-3-662-44757-4>
4. Bernstein, D.J.: Chacha, a variant of salsa20 (2008). <http://cr.yp.to/papers.html#chacha>
5. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O’Hearn, Z.: 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
6. Bernstein, D.J., Lange, T.: eBACS: Ecrypt benchmarking of cryptographic systems. <https://bench.cr.yp.to>. Accessed 11 May 2017
7. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: Keccak code package. <https://github.com/gvanas/KeccakCodePackage>. Accessed 02 May 2017
8. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V., Keer, R.V.: Kangarootwelve: fast hashing based on keccak-p. Cryptology ePrint Archive, Report 2016/770 (2016). <http://eprint.iacr.org/2016/770>
9. Buchmann, J., Dahmen, E., Hülsing, A.: XMSS - a practical forward secure signature scheme based on minimal security assumptions. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 117–129. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_8
10. Chang, D., Kumar, A., Morawiecki, P., Sanadhya, S.K.: 1st and 2nd preimage attacks on 7, 8 and 9 rounds of keccak-224,256,384,512. In: SHA-3 Workshop, August 2014
11. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, 30 October–03 November 2017, pp. 1825–1842. ACM (2017). <https://doi.org/10.1145/3133956.3133997>
12. Chen, M.-S., Hülsing, A., Rijneveld, J., Samardjiska, S., Schwabe, P.: From 5-Pass \mathcal{MQ} -based identification to \mathcal{MQ} -based signatures. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 135–165. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_5
13. Dahmen, E., Okeya, K., Takagi, T., Vuillaume, C.: Digital signatures out of second-preimage resistant hash functions. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 109–123. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88403-3_8
14. McGrew, D., Curcio, M., Fluhrer, S.: Hash-based signatures. <https://datatracker.ietf.org/doc/draft-mcgrew-hash-sigs/>. Accessed 22 May 2017

15. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - dilithium: Digital signatures from module lattices. IACR Cryptology ePrint Archive 2017, 633 (2017). <http://eprint.iacr.org/2017/633>
16. Espitau, T., Fouque, P.-A., Karpman, P.: Higher-order differential meet-in-the-middle preimage attacks on SHA-1 and BLAKE. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 683–701. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_33
17. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: fast-Fourier, lattice-based, compact signatures over NTRU. Submission to NIST Post-Quantum Competition (2017)
18. Goldreich, O.: The Foundations of Cryptography - Basic Applications, vol. 2. Cambridge University Press, Cambridge (2004)
19. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, pp. 212–219 (1996)
20. 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
21. Gueron, S., Mouha, N.: Sphincs-simpira: Fast stateless hash-based signatures with post-quantum security. Cryptology ePrint Archive, Report 2017/645 (2017). <http://eprint.iacr.org/2017/645>
22. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced meet-in-the-middle preimage attacks: first results on full tiger, and improved results on MD4 and SHA-2. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 56–75. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_4
23. Guo, J., Liu, M., Song, L.: Linear structures: applications to cryptanalysis of round-reduced Keccak. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 249–274. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_9
24. Hülsing, A.: W-OTS+ – shorter signatures for hash-based signature schemes. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 2013. LNCS, vol. 7918, pp. 173–188. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38553-7_10
25. Hülsing, A., Rijneveld, J., Schwabe, P.: ARMed SPHINCS. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 446–470. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_17
26. Jean, J.: Cryptanalysis of haraka. IACR Trans. Symmetric Cryptol. **2016**(1), 1–12 (2016)
27. Jean, J., Nikolić, I.: Efficient design strategies based on the AES round function. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 334–353. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_17
28. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: attacks on Skein-512 and the SHA-2 family. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 244–263. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_15
29. Kölbl, S., Lauridsen, M.M., Mendel, F., Rechberger, C.: Haraka v2 - efficient short-input hashing for post-quantum applications. IACR Trans. Symmetric Cryptol. **2016**(2), 1–29 (2016)

30. Leurent, G.: MD4 is not one-way. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 412–428. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71039-4_26
31. Reyzin, L., Reyzin, N.: Better than BIBA: short one-time signatures with fast signing and verifying. In: Batten, L., Seberry, J. (eds.) ACISP 2002. LNCS, vol. 2384, pp. 144–153. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45450-0_11
32. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: Ortiz, H. (ed.) Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, pp. 387–394. ACM (1990)
33. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**(5), 1484–1509 (1997)
34. Stephens, N., Biles, S., Boettcher, M., Eapen, J., Eyole, M., Gabrielli, G., Horsnell, M., Magklis, G., Martinez, A., Premillieu, N., et al.: The arm scalable vector extension. *IEEE Micro* **37**(2), 26–39 (2017)
35. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_1
36. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_2
37. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_2

Isogenies in Cryptography



Computing Isogenies Between Montgomery Curves Using the Action of $(0, 0)$

Joost Renes^(✉)

Digital Security Group, Radboud University, Nijmegen, The Netherlands
j.renes@cs.ru.nl

Abstract. A recent paper by Costello and Hisil at Asiacrypt'17 presents efficient formulas for computing isogenies with odd-degree cyclic kernels on Montgomery curves. We provide a constructive proof of a generalization of this theorem which shows the connection between the shape of the isogeny and the simple action of the point $(0, 0)$. This generalization removes the restriction of a cyclic kernel and allows for any separable isogeny whose kernel does not contain $(0, 0)$. As a particular case, we provide efficient formulas for 2-isogenies between Montgomery curves and show that these formulas can be used in isogeny-based cryptosystems without expensive square root computations and without knowledge of a special point of order 8. We also consider elliptic curves in triangular form containing an explicit point of order 3.

Keywords: Vélu's formulas · Montgomery form · 2-isogenies · SIDH Post-quantum cryptography

1 Introduction

Ever since their introduction to public-key cryptography by Miller [Mil86] and Koblitz [Kob87], elliptic curves have been of interest to the cryptographic community. By using the group of points on an appropriately chosen elliptic curve where the discrete logarithm problem is assumed to be hard, many standard protocols can be instantiated. Notably, the Diffie–Hellman key exchange [DH76] and the Schnorr signature scheme [Sch89] and its variants [Acc99, BDL+12] allow for efficient implementations with high security and small keys. The efficiency of these curve-based algorithms is largely determined by the scalar multiplication routine, and as a result a lot of research has gone into optimizing this operation.

However, the threat of large-scale quantum computers has initiated the search for alternative algorithms that also resist quantum adversaries (which the classical curve-based systems do not [Sho94]). Building on the work of Couveignes

J. Renes—This work has been supported by the Technology Foundation STW (project 13499 – TYPHOON & ASPASIA), from the Dutch government.

[Cou06] and Rostovsev and Stolbunov [RS06], in 2011 Jao and De Feo [JF11] proposed supersingular isogeny Diffie–Hellman (SIDH) as a key exchange protocol offering post-quantum security. Being based on the theory of elliptic curves, SIDH inherits several operations from traditional curve-based cryptography. As such, it has immediately benefited from decades of prior research into optimizing their operations. In particular, the Montgomery form of an elliptic curve has resulted in great performance. Initially proposed by Montgomery to speed up factoring using ECM [Mon87, Len87] and having been used for very efficient Diffie–Hellman key exchange (eg. Bernstein’s Curve25519 [Ber06]), the current fastest instantiations of SIDH also employ Montgomery curves [CLN16b, KAK16]. But, although the optimizations for scalar multiplication immediately carry over, the work on computing explicit isogenies on Montgomery curves is more limited.

For isogeny computations one commonly uses Vélu’s formulas [Vél71]. However, if the elliptic curve has a form which is less general than (or different from) Weierstrass form, the formulas from Vélu are not guaranteed to preserve this. As isogenies are only well-defined up to isomorphism, one can post-compose with an appropriate isomorphism to return to the required form, but it may not be obvious with which isomorphism, or the isomorphism may be expensive to compute. A more elegant approach is to observe some extra structure on the curve model and require the isogenies to preserve this. For example, Moody and Shumow [MS16] apply this idea to Edwards and Huff curves by fixing certain points. Moreover, since the isogeny is invariant under addition by kernel points, there is a close connection between the isogeny and the action (by translation) of some chosen point. We make this more explicit in Theorem 1 for curves in Weierstrass form.

So far the approaches for obtaining formulas for isogenies on Montgomery curves have been rather ad hoc. In [FJP14], De Feo, Jao and Plût apply Vélu’s formulas and compose with the appropriate isomorphisms to return to Montgomery form. As noted in [FJP14, Sect. 4.3.2], this approach fails to produce efficient results for 2-isogenies. That is, either one has to compute expensive square roots in a finite field (see eg. [CJL+17, Sect. 3.1]), or one relies on having an appropriate point of order 8. However, this point of order 8 is not readily available for the final two 2-isogenies. As one suggested workaround in [FJP14] they derive formulas for 4-isogenies between two curves in Montgomery form and propose to compute 2^e -isogenies as a chain of 4-isogenies. As a result, optimized SIDH implementations [CLN16a, KAK16] have employed curves where e is even so that 2^e -isogenies can be comprised entirely of 4-isogenies. In [CH17], Costello and Hisil present elegant formulas for isogenies between Montgomery curves, but their theorem covers only the case of odd cyclic kernels and subsequently also does not address the case of 2-isogenies. Moreover, there is no justification for the derivation of these isogenies (except for showing that they work).

We bridge this gap by providing a more thorough analysis on isogenies between Montgomery curves. We show that the isogenies arising in [CH17] are exactly those fixing $(0, 0)$. Since we enforce the isogeny to fix $(0, 0)$, this point cannot be in the kernel. We show in Proposition 1 that this is the only restric-

tion, and as a result present a generalization of [CH17, Theorem 1]. As a special case, we obtain formulas for 2-isogenies for 2-torsion points other than $(0, 0)$. We then show that this point can be naturally avoided in well-designed isogeny-based cryptosystems (see Sect. 4.3), and discuss the application of the 2-isogeny formulas to isogeny-based cryptosystems.

Finally, although currently it does not give rise to faster isogeny formulas, we consider it worthwhile to point out that the same techniques immediately apply to other models. In particular, models derived from the *Tate Normal Form* [Hus04, Sect. 4.4], where one could expect to get simple ℓ -isogenies for $\ell \geq 3$. We work out the case $\ell = 3$, also known as the *triangular form* [BCKL15], and derive isogenies by again fixing the action of the special point $(0, 0)$.

Organization. We begin by recalling some background on elliptic curves, isogenies and SIDH in Sect. 2. We state a theorem in Sect. 3 that allows to describe an isogeny in terms of the abscissas of its kernel points and their translations by a chosen point Q . We apply this to Montgomery curves in Sect. 4 and to curves in triangular form in Sect. 5, in both cases using $Q = (0, 0)$.

2 Preliminaries

An elliptic curve E defined over a field K is by definition [Gal12, Sil09] the curve

$$E/K : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3, \quad (1)$$

where $a_1, a_2, a_3, a_4, a_6 \in K$ such that E is non-singular. It is embedded into \mathbb{P}^2 with a single point $\mathcal{O}_E = (0 : 1 : 0)$ on the line $Z = 0$. This form is commonly referred to as *Weierstrass form* and the specified base point (implicitly) is \mathcal{O}_E . On the open patch defined by $Z \neq 0$ we can set $x = X/Z$ and $y = Y/Z$ and work on the corresponding affine curve inside \mathbb{A}^2 given by

$$E/K : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

We can move back to the projective curve by mapping $(x, y) \mapsto (x : y : 1)$. Therefore, although many equations are given in affine coordinates, they can easily be transformed into projective ones. For any extension L/K , the set of L -rational points $E(L)$ forms a group with identity \mathcal{O}_E [Sil09, Proposition 2.2(f)]. A subgroup $G \subset E(\bar{K})$ is said to be defined over K if $\sigma(P) \in G$ for all σ in the Galois group $\text{Gal}(\bar{K}/K)$.

Isogenies. Let E and \tilde{E} be elliptic curves. An isogeny ϕ from E to \tilde{E} is a surjective morphism such that $\phi(\mathcal{O}_E) = \mathcal{O}_{\tilde{E}}$ [Sil09, Sect. III.4]. The (finite) degree d of an isogeny is its degree as a morphism, and we say an isogeny is separable if $\#\ker(\phi) = d$. In this paper every isogeny that appears is assumed to be separable. Given a finite subgroup $G \subset E(\bar{K})$ defined over K , there exists a curve \tilde{E} and an isogeny $\phi : E \rightarrow \tilde{E}$ such that $\ker(\phi) = G$ [Gal12, Theorem 9.6.19]. The

curve \tilde{E} is unique up to isomorphism (over \bar{K}) and the isogeny ϕ is unique up to post-composition with an isomorphism. The isogeny ϕ can be made explicit by using Vélú’s formulas [Vél71] (for some fixed choice for the isogeny).

Montgomery Form. Setting $a_1 = a_3 = a_6 = 0$ and $a_4 = 1$ gives a curve in the form $E : y^2 = x^3 + ax^2 + x$. We also consider curves in the form $by^2 = x^3 + ax^2 + x$, better known as *Montgomery form*. Over \bar{K} these two curve forms are isomorphic via $(x, y) \mapsto (x, y\sqrt{b})$, but this isomorphism is only defined over K if $\sqrt{b} \in K$. In particular, if $K = \mathbb{F}_q$ and $\sqrt{b} \notin K$ then we call this curve a (non-trivial) quadratic twist. An easy check shows that $Q = (0, 0)$ is a K -rational point of order 2, while for any $Q_4 \in E(\bar{K})$ we have that $[2]Q_4 = Q$ if and only if

$$Q_4 \in \{(1, \pm\sqrt{(a+2)/b}), (-1, \pm\sqrt{(a-2)/b})\}$$

If P is any point of order 2 other than Q , then $x_P^2 + ax_P + 1 = 0$.

Tate Normal Form. Suppose we are given a curve E/K containing a point P of prime order $\ell \geq 3$. We can move P to $(0, 0)$ and its tangent line to the line $y = 0$. This transformation is completely K -rational and puts the curve in *Tate Normal Form* [Hus04, Sect. 4.4]

$$y^2 + axy + by = x^3 + cx^2, \quad a, b, c \in K.$$

In Sect. 5 we focus on the case where $\ell = 3$, in which case $c = 0$ and $b \neq 0$. Moreover, if $b = \beta^3$, then the transformation $(x, y) \mapsto (x/\beta^2, y/\beta^3)$ lets us assume that $b = 1$ and thus gives the form

$$E/K : y^2 + axy + y = x^3.$$

Note that β is not necessarily defined over K . However, Proposition 4 shows that once we start on such a curve, the 3-isogenies will preserve this form. It has discriminant $\Delta(E) = a^3 - 27$ and has a subgroup $\{\mathcal{O}_E, (0, 0), (0, -1)\}$ of order 3. The point $(0, 0)$ acts on points outside this subgroup by

$$(x, y) + (0, 0) = \left(\frac{-y}{x^2}, \frac{-y}{x^3} \right).$$

This curve is known as a *triangular curve* [BCKL15] and is isomorphic to the twisted Hessian curve [BCKL15, Theorem 5.3]

$$(a^3 - 27)x^3 + y^3 + 1 = 3axy.$$

SIDH. Let $e_A, e_B, f \in \mathbb{Z}_{\geq 0}$ such that $p = \ell_A^{e_A} \ell_B^{e_B} f - 1$ is prime. For $K = \mathbb{F}_{p^2}$ we can then find a supersingular curve E over K [Brö09] such that

$$\begin{aligned} \#E(K) &= (p + 1)^2, \\ E(K)[\ell_A^{e_A}] &= \mathbb{Z}/\ell_A^{e_A}\mathbb{Z} \times \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}, \\ E(K)[\ell_B^{e_B}] &= \mathbb{Z}/\ell_B^{e_B}\mathbb{Z} \times \mathbb{Z}/\ell_B^{e_B}\mathbb{Z}. \end{aligned}$$

By having the two parties compute isogenies of degree $\ell_A^{e_A}$ resp. $\ell_B^{e_B}$ and composing we can define a key exchange algorithm [FJP14, Sect. 3.2] similar to Diffie–Hellman. Since these degrees are exponentially large, they cannot be computed directly by polynomial evaluation. Instead, we decompose an $\ell_A^{e_A}$ -isogeny as a sequence of e_A isogenies of degree ℓ_A , which are efficiently computable for small ℓ_A [FJP14, Sect. 4] (typically $\ell_A \in \{2, 3\}$). Focusing on one of the sides, the secret key is a tuple $(\gamma, \delta) \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z} \times \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ where not both γ and δ are divisible by ℓ_A . Fixing a basis $E(K)[\ell_A^{e_A}] = \langle P, Q \rangle$, this corresponds to an isogeny with kernel $\langle [\gamma]P + [\delta]Q \rangle$. As the kernel is determined by its generator up to some invertible scalar multiple, and since at least one of the two scalars must be invertible, all keys can either be put in the form $(1, \delta)$ or $(\gamma, 1)$.

3 Isogenies on Weierstrass Curves

We begin by stating a straightforward, but rather useful theorem. By assuming to have knowledge on the action of an isogeny on a single point Q , we can translate this point by elements of the kernel to obtain a simple description of the isogeny. Many curve models have a natural choice for this point (eg. $Q = (0, 0)$ in Montgomery form, see Sect. 4).

Theorem 1. *Let K be a field and E/K an elliptic curve in Weierstrass form. Let $G \subset E(\bar{K})$ be a finite subgroup defined over K and*

$$\phi : (x, y) \mapsto (f(x), c_0 y f'(x) + g(x)), \quad c_0 \in \bar{K}^*, \quad (2)$$

a separable isogeny such that $\ker(\phi) = G$. Let $Q \in E(\bar{K})$ such that $Q \notin G$. Then

$$f(x) = c_1(x - x_Q) \prod_{T \in G \setminus \{\mathcal{O}_E\}} \frac{(x - x_{Q+T})}{(x - x_T)} + f(x_Q), \quad \text{for } c_1 \in \bar{K}^*.$$

Proof. First note that the existence of ϕ follows from Vélú’s formulas [Vél71], while a standard result [Gal12, Theorem 9.7.5] shows that it can be written in the form of (2) (where $f'(x)$ is the formal derivative df/dx of $f(x)$). More explicitly, following the notation of [Gal12, Theorem 25.1.6], there exist functions $u, t : G \setminus \{\mathcal{O}_E\} \rightarrow \bar{K}$ such that

$$f(x) = x + \sum_{T \in G_1 \cup G_2} \left(\frac{t(T)}{x - x_T} + \frac{u(T)}{(x - x_T)^2} \right),$$

where $G_2 \subset G$ is the set of points of order 2 and $G_1 \subset E(\bar{K})$ is such that

$$G = \{\mathcal{O}_E\} \cup G_2 \cup G_1 \cup \{-T : T \in G_1\}.$$

Moreover, $u(T) = 0$ if and only if T has order 2. Collecting denominators, it is then immediate that there exists a function $w \in \bar{K}[x]$ such that $\deg(w) = |G|$ and

$$f(x) = \frac{w(x)}{v(x)}, \quad \text{where } v(x) = \prod_{T \in G \setminus \{\mathcal{O}_E\}} (x - x_T).$$

Now define

$$h(x) = w(x)v(x_Q) - w(x_Q)v(x).$$

Note that clearly $h(x_Q) = 0$. Since the value of f is invariant under the action of points in G , we in fact have that $h(x_{Q+T}) = 0$ for all $T \in G$. Therefore it follows that $(x - x_{Q+T}) \mid h(x)$ for all $T \in G$. If for all $T_1, T_2 \in G$ such that $T_1 \neq T_2$ we have that $x_{Q+T_1} \neq x_{Q+T_2}$, then it immediately follows that

$$\prod_{T \in G} (x - x_{Q+T}) \mid h(x).$$

Otherwise¹, assume we have $T_1, T_2 \in G$ such that $T_1 \neq T_2$ and $x_{Q+T_1} = x_{Q+T_2}$. Since any x -coordinate corresponds to at most 2 points on E , it follows that $Q + T_2 = \pm(Q + T_1)$. However, $Q + T_2 = Q + T_1$ implies that $T_1 = T_2$, which contradicts our assumption. Therefore $Q + T_2 = -(Q + T_1)$ and

$$\begin{aligned} [2]\phi(Q + T_1) &= \phi(Q + T_1) + \phi(Q + T_1) \\ &= \phi(Q + T_1) + \phi(Q + T_2) \\ &= \phi(Q + T_1) - \phi(Q + T_1) \\ &= \mathcal{O}_{\bar{E}}. \end{aligned}$$

Moreover,

$$\begin{aligned} [2](Q + T_1) = \mathcal{O}_E &\iff Q + T_1 + Q + T_1 = \mathcal{O}_E \\ &\iff Q + T_1 - (Q + T_2) = \mathcal{O}_E \\ &\iff T_1 = T_2, \end{aligned}$$

which contradicts the assumption that $T_1 \neq T_2$. Thus $\psi_2(Q + T_1) \neq 0$ and by Lemma 1 we can conclude that $f'(x_{Q+T_1}) = 0$. Since away from the zeros of v we have

$$h(x) = (f(x) - f(x_Q))v(x)v(x_Q),$$

it follows from the fact that $f'(x_{Q+T_1}) = f(x_{Q+T_1}) - f(x_Q) = 0$ that $h'(x_{Q+T_1}) = 0$. That is, h has (at least) a double root at x_{Q+T_1} . In other words,

$$(x - x_{Q+T_1})(x - x_{Q+T_2}) \mid h(x).$$

It is then clear that indeed

$$\prod_{T \in G} (x - x_{Q+T}) \mid h(x).$$

¹ This proof is quite elementary. An alternative method (which is perhaps more illuminating) is to consider the divisor of $x - f(x_Q)$ on E/G and to pull it back via ϕ . We can then use the fact that $\text{div}(f(x) - f(x_Q)) = \phi^* \text{div}(x - f(x_Q))$.

As $\deg(h) \leq \max(\deg(w), \deg(v)) = |G|$, there exists a constant $c \in K^*$ such that

$$h(x) = c \prod_{T \in G} (x - x_{Q+T}).$$

Thus,

$$f(x) = \frac{w(x)}{v(x)} = \frac{h(x)}{v(x)v(x_Q)} + f(x_Q).$$

The result follows by setting $c_1 = c/v(x_Q)$. □

Lemma 1. *Let the setup be as in Theorem 1 and let $R \in E(\bar{K}) \setminus G$. Then*

$$[2]\phi(R) = \mathcal{O}_{\tilde{E}} \iff \psi_2(R)f'(x_R) = 0,$$

where ψ_2 is the 2-division polynomial.

Proof. Firstly note that $R \notin G$ and thus $\phi(R) \neq \mathcal{O}_{\tilde{E}}$. Therefore, by definition of the 2-division polynomial on $\tilde{E} = E/G$ it follows that

$$[2]\phi(R) = \mathcal{O}_{\tilde{E}} \iff 2y_{\phi(R)} + \tilde{a}_1x_{\phi(R)} + \tilde{a}_3 = 0,$$

where \tilde{a}_1 and \tilde{a}_3 are Weierstrass constants of \tilde{E} conform (1). Using the definition of ϕ and by recalling that (see eg. [Gal12, Theorem 9.7.5])

$$2g(x_R) = c_0(a_1x_R + a_3)f'(x_R) - \tilde{a}_1f(x_R) - \tilde{a}_3,$$

a straightforward computation shows that

$$2y_{\phi(R)} + \tilde{a}_1x_{\phi(R)} + \tilde{a}_3 = 0 \iff c_0(2y_R + a_1x_R + a_3)f'(x_R) = 0.$$

Finally observe that $\psi_2(R) = 2y_R + a_1x_R + a_3$ and $c_0 \neq 0$. □

Remark 1. Theorem 1 shows the connection between ϕ and the action of the point Q on abscissas of kernel elements, as ϕ is given by a product of functions

$$\frac{x - x_{Q+T}}{x - x_T}.$$

If this action is simple (eg. in Montgomery form where $x_{(0,0)+T} = 1/x_T$) then we can expect simple formulas for isogenies.

Remark 2. By relying on Theorem 1 we simplify the proof compared to earlier works [CH17, MS16]. Whereas those works present rational maps and prove them to be isogenies, we turn this argument around. We use the existence of the isogeny (by Vélú’s formulas) and apply appropriate isomorphisms to enforce some structure to be maintained (eg. $(0, 0) \mapsto (0, 0)$ in Montgomery form). We can then apply Theorem 1 to get formulas for the isogeny up to some constants. Finally we also use the formal group law. However, as opposed to proving that the rational functions defining the isogeny satisfy the curve relation of the co-domain curve, we can assume them to vanish and therefore extract the constants and the coefficients of the co-domain curve. This significantly simplifies the proof compared to earlier works (eg. [MS16, Theorem 2] and [CH17, Theorem 1]).

4 Montgomery Form and 2-Isogenies

In [CH17, Theorem 1] Costello and Hisil present rational maps which they prove to be isogenies between Montgomery curves. These isogenies are not unique, and are for example different from the formulas directly derived using Vélú’s formulas. It is immediate that the isogenies in [CH17] have the property of fixing $(0, 0)$. In Sect. 4.1 we show that this fact, together with the co-domain curve being in Montgomery form, characterizes their formulas (up to some sign choices). This generalizes the theorem by Costello and Hisil, by removing the restriction of kernels being cyclic and having odd order. In particular, in Sect. 4.2 we present formulas for 2-isogenies determined by points of order 2 other than $(0, 0)$. Until now these had not appeared, and were considered to require the computation of a square root. In Sect. 4.3 we show how one could apply these formulas in an implementation. Although it requires only a modest change to the parameters, this does require care and can simplify the implementation. Finally in Sect. 4.4 we comment on a comparison to the state-of-the-art.

4.1 The General Formula

We begin by stating Proposition 1, which is the analogue of [CH17, Theorem 1].

Proposition 1. *Let K be a field with $\text{char}(K) \neq 2$. Let $a \in K$ such that $a^2 \neq 4$ and $E/K : y^2 = x^3 + ax^2 + x$ is a Montgomery curve. Let $G \subset E(\bar{K})$ be a finite subgroup such that $(0, 0) \notin G$ and let ϕ be a separable isogeny such that $\ker(\phi) = G$. Then there exists a curve $\tilde{E}/K : y^2 = x^3 + Ax^2 + x$ such that, up to post-composition by an isomorphism,*

$$\begin{aligned} \phi : E &\rightarrow \tilde{E} \\ (x, y) &\mapsto (f(x), c_0 y f'(x)) \end{aligned}$$

where

$$f(x) = x \prod_{T \in G \setminus \{\mathcal{O}_E\}} \frac{xx_T - 1}{x - x_T}.$$

Moreover, writing

$$\pi = \prod_{T \in G \setminus \{\mathcal{O}_E\}} x_T, \quad \sigma = \sum_{T \in G \setminus \{\mathcal{O}_E\}} \left(x_T - \frac{1}{x_T} \right),$$

we have that $A = \pi(a - 3\sigma)$ and $c_0^2 = \pi$.

Proof. Over \bar{K} we can always move E/G to Montgomery form. Let $P \in E(\bar{K})$ such that $x_P = 1$. Then $[2]P = (0, 0)$, hence $[2]\phi(P) = \phi([2]P) \neq \mathcal{O}_{E/G}$ while $[4]\phi(P) = [2](0, 0) = \mathcal{O}_{E/G}$. Thus $\phi(P)$ is a point of exact order 4, and we apply an isomorphism such that $x_{\phi(P)} = (-1)^{|G|-1}$ (see eg. [FJP14, Sect. 4.3.2]). In

particular this assures that $\phi : (0, 0) \mapsto (0, 0)$. We then twist the y -coordinate via another isomorphism to set the coefficient of y^2 to 1 and have

$$\tilde{E} = E/G : y^2 = x^3 + Ax^2 + x.$$

Now apply Theorem 1 with $Q = (0, 0)$. We obtain that

$$\begin{aligned} f(x) &= c_1(x - x_{(0,0)}) \prod_{T \in G \setminus \{\mathcal{O}_E\}} \frac{(x - x_{(0,0)+T})}{(x - x_T)} + f(x_{(0,0)}) \\ &= c_1 x \prod_{T \in G \setminus \{\mathcal{O}_E\}} \frac{(x - \frac{1}{x_T})}{(x - x_T)}. \end{aligned}$$

As we set up \tilde{E} such that $f(1) = (-1)^{|G|-1}$, we find that

$$c_1 = \prod_{T \in G \setminus \{\mathcal{O}_E\}} x_T.$$

Feeding c_1 back into the equation for f puts it in the right form. At this point it only remains to find A and c_0 (observe that $g = 0$ in Montgomery form [Gal12, Theorem 9.7.5]). To this end we utilize the formal group law, similar to [CH17, MS16].

Let $t = x/y$ be a uniformizer at \mathcal{O}_E and write $s = 1/y$. By observing that $s = t^3 + at^2s + ts^2$ we can recursively substitute s into itself to get an expression $s(t) \in \mathbb{Z}[a][[t]]$ as a power series²

$$s(t) = t^3 + at^5 + (a^2 + 1)t^7 + O(t^9)$$

This is well-defined, see for example [Sil09, Sect. IV.1]. As a result we can write

$$\begin{aligned} 1/s(t) &= y(t) = t^{-3} - at^{-1} + O(t), \\ ty(t) &= x(t) = t^{-2} - a + O(t^2). \end{aligned}$$

Let $X(t) = f(x(t))$. Then

$$\begin{aligned} X(t) &= \pi t^{-2} + \pi(\sigma - a) + O(t^2), \\ dX/dt &= -2\pi t^{-3} + O(t), \\ dx/dt &= -2t^{-3} + O(t), \\ (dx/dt)^{-1} &= -t^3/2 + O(t^7). \end{aligned}$$

Now define

$$\begin{aligned} Y(t) &= c_0 y(t) \cdot (df/dx) \\ &= c_0 y(t) \cdot (dX/dt) \cdot (dx/dt)^{-1}, \end{aligned}$$

² We denote by $O(t^n)$ a series whose coefficients of t^m are zero for all $m < n$.

so that

$$Y(t) = c_0\pi t^{-3} - c_0a\pi t^{-1} + O(t).$$

Writing

$$F(t) = Y(t)^2 - (X(t)^3 + AX(t)^2 + X(t))$$

it follows that

$$F(t) = F_{-6} \cdot t^{-6} + F_{-4} \cdot t^{-4} + O(t^{-2}),$$

with

$$F_{-6} = \pi^2(c_0^2 - \pi), \quad F_{-4} = \pi^2(3\pi(a - \sigma) - 2ac_0^2 - A).$$

Now since by assumption ϕ is an isogeny with co-domain curve \tilde{E} , and since F is precisely the equation defining \tilde{E} , we must have $F = 0$. Solving $F_{-6} = 0$ and $F_{-4} = 0$ simultaneously leads to the desired equations for c_0^2 and A . Note that this way we have only defined c_0 up to sign. However, the sign choice merely induces a composition with $[-1]$ and therefore does not affect ϕ up to isomorphism. \square

Remark 3. It is perhaps not immediately obvious that Proposition 1 is a generalization of the result by Costello and Hisil [CH17, Theorem 1]. Our result assumes the domain curve E to be of the form $y^2 = x^3 + ax^2 + x$, while their theorem also accounts for curves $E_0 : by^2 = x^3 + ax^2 + x$. Moreover, the map itself looks slightly different. However, it is straightforward to check that if one pre-composes with the isomorphism

$$\begin{aligned} \psi_0 : E_0 &\rightarrow E \\ (x, y) &\mapsto (x, y\sqrt{b}) \end{aligned}$$

and post-composes with the isomorphism

$$\begin{aligned} \psi_1 : \tilde{E} &\rightarrow E_1 : By^2 + x^3 + Ax^2 + x, \\ (x, y) &\mapsto \left(x, \frac{y}{\sqrt{\pi b}}\right) \end{aligned}$$

then one recovers the theorem from Costello and Hisil in the case of odd-degree cyclic kernels. Ignoring these twists in Proposition 1 simplifies the proof. For example, see Proposition 2.

Remark 4. If $K = \mathbb{F}_q$ is a (large-characteristic) finite field, then possibly π is a non-square in \mathbb{F}_q . As a result ϕ is not defined over \mathbb{F}_q . However, in that case the map

$$(x, y) \mapsto (f(x), yf'(x))$$

is defined over \mathbb{F}_q with co-domain curve $\tilde{E}^{(t)} : \pi y^2 = x^3 + Ax^2 + x$. This is the quadratic twist of \tilde{E} . Since \tilde{E} and its twist have the same Kummer line, we eliminate this issue by projecting to \mathbb{P}^1 (ie. by using x -only arithmetic).

Remark 5. If we set up an SIDH instance with $\ell_A = 2$ and $e_A \geq 2$ then the x -coordinates of points of order 2 are in fact squares. This follows from [Hus04, Chap. 1, Theorem 4.1] combined with the doubling formulas for Montgomery curves, as noted in [CJL+17, Sect. 3.2]. Since all x -coordinates of points with orders other than 2 appear twice in the equation for π , it follows that π is actually a square. Therefore ϕ is defined over \mathbb{F}_{p^2} , and in particular we always have $\#E(\mathbb{F}_{p^2}) = \#\tilde{E}(\mathbb{F}_{p^2})$. This is (implicitly) used in formulas for public-key compression [CJL+17, ZJP+17].

4.2 2-Isogenies

As an immediate consequence of Proposition 1 we obtain formulas for 2-isogenies for 2-torsion points other than $(0, 0)$.

Proposition 2. *Let K be a field with $\text{char}(K) \neq 2$. Let $a, b \in K$ such that $b \neq 0$ and $a^2 \neq 4$, and $E/K : by^2 = x^3 + ax^2 + x$ is a Montgomery curve. Let $P \in E(\bar{K})$ such that $P \neq (0, 0)$ and $[2]P = \mathcal{O}_E$. Then*

$$\begin{aligned} \phi : E &\rightarrow \tilde{E}/K : By^2 = x^3 + Ax^2 + x \\ (x, y) &\mapsto (f(x), yf'(x)), \end{aligned}$$

with $B = x_P b$ and $A = 2(1 - 2x_P^2)$ is a 2-isogeny with $\ker(\phi) = \langle P \rangle$, where

$$f(x) = x \cdot \frac{xx_P - 1}{x - x_P}.$$

Proof. This is exactly the statement in Proposition 1 composed with the isomorphisms ψ_0 and ψ_1 from Remark 3. The result follows by using the identity $ax_P = -(x_P^2 + 1)$ to derive A . □

We also compute the kernel of the dual of ϕ , which will be helpful in Sect. 4.3 for larger degree isogenies.

Corollary 1. *Let the setup be as in Proposition 2. Then $\ker(\hat{\phi}) = \langle (0, 0) \rangle$.*

Proof. Let ψ be a separable isogeny with domain \tilde{E} and kernel $\langle (0, 0) \rangle$. Then certainly $E[2] \subset \ker(\psi \circ \phi)$, and since $\deg(\psi \circ \phi) = 4$ we in fact have $E[2] = \ker(\psi \circ \phi)$. Thus $\psi = \hat{\phi}$ up to isomorphism by uniqueness of the dual isogeny, and hence $\ker(\hat{\phi}) = \ker(\psi)$. □

The statement and proof of Proposition 2 does not explain why we are able to compute 2-isogenies without explicit square roots, while earlier works [CH17, FJP14] could not. We provide a more direct computation in Remark 6 to show why this is the case.

Remark 6. In [FJP14, Sect. 4.3.2] the authors describe a 2-isogeny with kernel $(0,0)$ as

$$\begin{aligned} \varphi : E &\rightarrow F : by^2 = x^3 + (a + 6)x^2 + 4(2 + a)x \\ (x, y) &\mapsto \left(\frac{(x - 1)^2}{x}, y \left(1 - \frac{1}{x^2} \right) \right). \end{aligned}$$

The coefficient of x can be removed by computing $2\sqrt{a + 2}$ and composing with the isomorphism

$$(x, y) \mapsto \left(\frac{x}{2\sqrt{a + 2}}, \frac{y}{2\sqrt{a + 2}} \right),$$

putting F in the desired form. This requires computing a square root, which could be avoided by having knowledge of a point $P_8 = (2\sqrt{a + 2}, -)$ of order 8 above $(0, 0)$. Instead, we observe that we can compose with the isomorphism

$$\begin{aligned} \psi : F &\rightarrow G : \frac{b}{\sqrt{a^2 - 4}}y^2 = x^3 - \frac{2a}{\sqrt{a^2 - 4}} \cdot x^2 + x \\ (x, y) &\mapsto \left(\frac{x + a + 2}{\sqrt{a^2 - 4}}, \frac{y}{\sqrt{a^2 - 4}} \right), \end{aligned}$$

which moves the kernel of $\tilde{\varphi}$ to $(0,0)$. This requires computing $\sqrt{a^2 - 4}$ and therefore also relies on a square root. However, if $P_2 = (x_2, 0)$ is a point of order 2 on E with $x_2 \neq 0$, then $x_2^2 + ax_2 + 1 = 0$. Therefore it is immediate that

$$\sqrt{a^2 - 4} = 2x_2 + a,$$

allowing us to compute the isomorphism efficiently. We have such a point by assumption in Proposition 2. We can now compute ϕ as $\psi \circ \varphi \circ \chi$, where χ is an isomorphism mapping P_2 to $(0, 0)$ (eg. [FJP14, Eq. (15)]).

To provide explicit operation counts³ we move to projective space and project to \mathbb{P}^1 . Let $P = (X_P : 0 : Z_P)$ be a point of order 2 on $E : bY^2Z = X^3 + aX^2Z + XZ^2$ such that $X_P \neq 0$. Then by Proposition 2

$$\begin{aligned} \phi : E &\rightarrow \tilde{E} : BY^2Z = X^3 + AX^2Z + XZ^2 \\ (X : - : Z) &\mapsto (X(XX_P - ZZ_P) : - : Z(XZ_P - ZX_P)) \end{aligned}$$

is a 2-isogeny with kernel $\langle P \rangle$. We have

$$A = 2(Z_P^2 - 2X_P^2)/Z_P^2,$$

and to avoid inversions we represent it projectively as

$$(A : 1) = (2(Z_P^2 - 2X_P^2) : Z_P^2).$$

³ We denote by \mathbf{M} , \mathbf{S} resp. \mathbf{a} the cost of a field multiplication, squaring resp. addition or subtraction (which are assumed to have equal cost).

However, the doubling formulas on Montgomery curves use $(A+2)/4$ instead of A , and we see that

$$(A + 2 : 4) = (-X_P^2 : Z_P^2).$$

This can be computed in $2\mathbf{S} + 1\mathbf{a}$, but one can easily integrate the negation into the doubling formulas to reduce the cost to $2\mathbf{S}$. Moreover, we observe that

$$\begin{aligned} X(XX_P - ZZ_P) &= X \left[(X - Z)(X_P + Z_P) + (X + Z)(X_P - Z_P) \right], \\ Z(XZ_P - ZX_P) &= Z \left[(X - Z)(X_P + Z_P) - (X + Z)(X_P - Z_P) \right]. \end{aligned}$$

This can be computed in $4\mathbf{M} + 6\mathbf{a}$ via the sequence of operations

$$\begin{aligned} T_0 &= X_P + Z_P, T_1 = X_P - Z_P, T_2 = X + Z, T_3 = X - Z, T_4 = T_3 \cdot T_0, \\ T_5 &= T_2 \cdot T_1, T_6 = T_4 + T_5, T_7 = T_4 - T_5, T_8 = X \cdot T_6, T_9 = Z \cdot T_7. \end{aligned}$$

If we assume $X_P + Z_P$ and $X_P - Z_P$ to be pre-computed, the cost reduces to $4\mathbf{M} + 4\mathbf{a}$. This would for example apply if we require multiple evaluations of the isogeny (eg. in SIDH).

4.3 Application to Isogeny-Based Cryptography

In the general setting it is not true that the kernels appearing in the computations cannot contain the point $(0, 0)$, so it is not clear that the 2-isogenies can immediately be used. In a similar fashion, it is not true in general that kernels of 4-isogenies cannot contain $(1, \pm\sqrt{(a+2)/b})$ or $(-1, \pm\sqrt{(a-2)/b})$. In [CLN16a, Sect. 3] and [CH17] this assumption is used without justification (implicitly by replacing ψ_4 with $\hat{\psi}_4$). This is dealt with by using a separate function `first_4_isog` for the first 4-isogeny, which is the only kernel that can contain such a point (a proof of which does not appear). However, Lemma 2 and Corollary 2 show that we can avoid these points with only a minor restriction on the keyspace. Applying this restriction to [CLN16a] makes the function `first_4_isog` redundant, simplifying the implementation.

Lemma 2. *Let $e, f \in \mathbb{Z}_{\geq 0}$ and let $p = 2^e 3^f - 1$ be prime. Let E/\mathbb{F}_{p^2} be a supersingular elliptic curve in Montgomery form such that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. Let $P, Q \in E(\mathbb{F}_{p^2})$ such that $E[2^e] = \langle P, Q \rangle$ and $[2^{e-1}]Q = (0, 0)$. Let $\alpha \in \mathbb{Z}_{2^e}$. Then $(0, 0) \notin \langle P + [\alpha]Q \rangle$.*

Proof. It is clear that $\langle P + [\alpha]Q \rangle$ can only contain a single point of order 2, namely $[2^{e-1}](P + [\alpha]Q)$. But by assumption on Q we know that $[2^{e-1}](P + [\alpha]Q) \neq (0, 0)$, hence the result follows.

By Lemma 2 we know that we can compute the 2^e -isogenies as defined in Proposition 1. However, as the degrees grow this will quickly be impractical. Instead, we do the computations as a sequence of 2-isogenies (ie. as in Proposition 2) [FJP14, Sect. 4]. Therefore we must show that none of these intermediate isogenies has a kernel generated by $(0, 0)$.

Corollary 2. *Let the setup be as in Lemma 2 and write $R = P + [\alpha]Q$. Let ϕ be an isogeny such that $\ker(\phi) = \langle R \rangle$ and suppose that we compute*

$$\begin{aligned} \phi &= \phi_{e-1} \circ \dots \circ \phi_0, \\ \ker(\phi_0) &= \langle [2^{e-1}]R \rangle, \\ \ker(\phi_i) &= \langle [2^{e-i-1}] \phi_{i-1} \dots \phi_0(R) \rangle, \quad (\text{for } 1 \leq i \leq e-1) \end{aligned}$$

as a sequence of 2-isogenies, each one computed as in Proposition 2. Then $(0, 0) \notin \ker(\phi_i)$ for all $0 \leq i \leq e-1$.

Proof. We apply induction on i . The statement for $i = 0$ follows from Lemma 2. Let $i > 0$. Then $\ker(\widehat{\phi}_{i-1}) = \langle (0, 0) \rangle$ by the inductive hypothesis and by Corollary 1. But since the walk determined by ϕ is non-backtracking, it follows that $\ker(\phi_i) \neq \langle (0, 0) \rangle$. As $\#\ker(\phi_i) = 2$, we conclude that $(0, 0) \notin \ker(\phi_i)$.

The keyspace is determined by tuples (γ, δ) which define kernels of the form $\langle [\gamma]P + [\delta]Q \rangle$, where not simultaneously $\gamma \equiv 0 \pmod 2$ and $\delta \equiv 0 \pmod 2$. We can divide the space into the three disjoint sets (of equal size)

$$\mathcal{K}_{(i,j)} = \{(\gamma, \delta) : \gamma \equiv i \pmod 2, \delta \equiv j \pmod 2\},$$

for $(i, j) \in \{(0, 1), (1, 0), (1, 1)\}$. The restriction on the keyspace then corresponds exactly to disallowing $\mathcal{K}_{(0,1)}$, removing 1/3 of the keyspace. It is easy to see that these keys define the isogeny walks for which the first 2-isogeny has kernel $\langle (0, 0) \rangle$. Note that this depends on the choice of 2^e -torsion basis $\{P, Q\}$, where we choose Q to lie above $(0, 0)$. A similar argument applies to the use of 4-isogenies in [CLN16a].

Remark 7. The initial proposal to use curves in Montgomery form [CLN16a, Sect. 4] suggested taking P as an \mathbb{F}_p -rational point on the curve $E_0/\mathbb{F}_p : y^2 = x^3 + x$ with $j(E_0) = 1728$ and Q as the image of P under the distortion map $(x, y) \mapsto (-x, iy)$. This allows a compressed representation of $\{P, Q\}$. Although this does not work for the basis as chosen in Lemma 2, it only results in a small increase in the size of public parameters (which never need to be transferred).

4.4 Relating 2-Isogenies and 4-Isogenies

It is easy to see that the 4-isogenies from [CH17, Appendix A], which are currently the fastest formulas, can be derived by applying the 2-isogenies from Sect. 4.2 twice. That is, since they have equal kernel they are equal up to composition with an isomorphism. Both isogenies have a Montgomery curve as co-domain, of which there are at most six per isomorphism class (by looking at the formula for the j -invariant). Also, in both cases the dual is generated by a point $P \in \{(1, \pm\sqrt{(a+2)/b}), (-1, \pm\sqrt{(a-2)/b})\}$. Therefore we can transform one into the other by possibly composing with the simple isomorphisms $(x, y) \mapsto (x, -y)$ and $(x, y) \mapsto (-x, iy)$, where $i \in \bar{K}$ such that $i^2 = -1$. As a result, applying the 2-isogenies twice will not have more efficient formulas than the 4-isogenies. Indeed, if this were the case we could use the above transformation to obtain equally fast 4-isogenies. We summarize the costs in Table 1.

Table 1. Comparison of the costs of evaluating 2-isogenies and 4-isogenies.

Operation	2-isogeny	2×2 -isogeny	4-isogeny [CH17]
Compute $(A + 2 : 4)$	2S	4S	4S + 5a ^a
First evaluation	4M + 6a	8M + 12a	6M + 2S + 6a
Subsequent evaluations	4M + 4a	8M + 8a	6M + 2S + 6a

^aMany of these additions are not needed to compute $(A + 2 : 4)$, but are used as pre-computation for the isogeny evaluation. We provide the counts as is to align with [CH17] since it does not affect our comparison of the costs of large degree isogeny evaluations.

Besides their theoretic value, there are some small upsides to using 2-isogenies in an implementation. Firstly, the computation leaks only a single bit as opposed to two [FJP14, Sect. 4.3.2]. Instead of leaking the dual of the final 4-isogeny, it would only leak the dual of the last 2-isogeny. Also, in some cases one may be able to select smaller parameters for a certain given security level. Primes of the form $2^e 3^f - 1$ where $e \approx \log_2(3^f)$ are somewhat sparse, and depending on one’s requirements restricting e to be even could result in a (much) larger prime than hoped for. Alternatively, one could of course achieve this by doing a single 2-isogeny followed by a chain of 4-isogenies. However, this does come at the cost of having to implement more algorithms, increasing the size and complexity of an (already complex) implementation. Finally, having worked out formulas for isogenies of even degree and by showing how to avoid $(0, 0)$, we are able to straightforwardly write down formulas for 2^e -isogenies with $e \geq 3$. It remains to be seen if these can be made more efficient than repeated applications of 4-isogenies.

5 Triangular Form and 3-Isogenies

Given the generality of Theorem 1, an obvious question is whether there are other classes of curves which could possibly give rise to simple formulas for isogenies. In this section we analyze curves in triangular form $E/K : y^2 + axy + y = x^3$ containing a point $(0, 0)$ of order 3. Most of the ideas from earlier sections apply and in particular we get analogous statements for computing 3-isogenies (see Sect. 5.2). Although these allow to compute the co-domain curve very efficiently, the evaluation of the isogeny is not as efficient as its Montgomery counterpart. Moreover, since tripling formulas are currently slower, at this point Montgomery form still performs better with respect to 3-isogenies.

5.1 The General Formula

We start by presenting formulas for triangular curves that work for any separable isogeny whose kernel is an odd order subgroup. It is possible to include groups of even order, but this creates a case distinction which makes the proof more

tedious. Since having (enough) rational points of even order would enable us to go to Montgomery form and reduce to Sect. 4, we discard that case here.

There are a couple of (minor) complications compared to the proof of Proposition 1. Firstly, we cannot assume that $g = 0$. If we work on \mathbb{P}^1 this will not affect the efficiency, but we will have to take it into account in the proof. Secondly, the action of $(0, 0)$ does not involve only x -coordinates. To eliminate the y -coordinates that arise, we group the kernel points into sets $\{T, -T\}$ (similar to [CH17, Theorem 1]).

Proposition 3. *Let K be a field with $\text{char}(K) \neq 2$. Let $a \in K$ such that $a^3 \neq 27$ and $E/K : y^2 + axy + y = x^3$ in triangular form. Let $G \subset E(\bar{K})$ be a finite subgroup of odd order such that $(0, 0) \notin G$ and let ϕ be a separable isogeny such that $\ker(\phi) = G$. Let*

$$X = \left\{ x_P \mid P \in G \setminus \{O_E\} \right\}.$$

Then there exists a curve $\tilde{E}/K : y^2 + Axy + y = x^3$ such that, up to post-composition by an isomorphism,

$$\begin{aligned} \phi : E &\rightarrow \tilde{E} \\ (x, y) &\mapsto (f(x), c_0 y f'(x) + g(x)) \end{aligned}$$

where

$$f(x) = x \prod_{z \in X} \frac{x^2 z^2 - x(az + 1) - z}{(x - z)^2}.$$

Moreover, writing

$$\pi = \prod_{z \in X} z, \quad \sigma = \sum_{z \in X} \left(\frac{1}{z^2} + \frac{a}{z} - 2z \right),$$

we have that $A^2 = \pi^2(a^2 + 12\sigma)$ and $c_0 = (-1)^{|X|}\pi$.

Proof (sketch). This proof is almost completely analogous to the one of Proposition 1. That is, we put E/G in triangular form by moving the image of $(0, 0)$ under ϕ to $(0, 0)$. We then apply Theorem 1 with $Q = (0, 0)$ and use that

$$y_T y_{-T} = -x_T^3, \quad \text{and} \quad y_T + y_{-T} = -ax_T - 1.$$

By observing that $\phi : (0, -1) \mapsto (0, -1)$ we find that

$$c_0 = (-1)^{|X|} \pi^3 / c_1,$$

and finally we apply the formal group law to find expressions for c_1 and A . \square

5.2 3-Isogenies

We work out explicit formulas for 3-isogenies.

Proposition 4. *Let K be a field with $\text{char}(K) \neq 2$. Let $a \in K$ such that $a^3 \neq 27$ and $E/K : y^2 + axy + y = x^3$ in triangular form. Let $P \in E(\bar{K})$ a point such that $[3]P = \mathcal{O}_E$ and $x_P \neq 0$. Then*

$$\begin{aligned} \phi : E &\rightarrow \tilde{E}/K : y^2 + Axy + y = x^3 \\ (x, y) &\mapsto (f(x), -x_P y f'(x) + g(x)) \end{aligned}$$

with $A = -3(2 + ax_P)$ is a 3-isogeny such that $\ker(\phi) = \langle P \rangle$, where

$$f(x) = x \cdot \frac{x^2 x_P^2 - x(ax_P + 1) - x_P}{(x - x_P)^2}.$$

Proof. This is Proposition 3 with $X = \{x_P\}$. Using the division polynomial

$$\psi_3(x) = x(3x^3 + a^2x^2 + 3ax + 3)$$

it follows that $9(2+ax_P)^2 = \pi^2(a^2 + 12\sigma)$. Hence $A = \pm 3(2+ax_P)$ and the only remaining uncertainty is the choice of sign. However, setting $A = -3(2 + ax_P)$, a direct computation shows that

$$f'(x) = x_P^2 \cdot \frac{((x - x_P)^3 - (6x_P^2 + a^2x_P + a)x + x_P^3 + 1)}{(x - x_P)^3},$$

while

$$g(x) = x^3 \cdot \frac{((3 + ax_P)x_P^2x + x_P^3 + 1)}{(x - x_P)^3}.$$

For $X = f(x)$ and $Y = -x_P y f'(x) + g(x)$, a straightforward calculation shows that $Y^2 + AX Y + Y = X^3$. It is then clear that ϕ is an isogeny and that $\ker(\phi) = \langle P \rangle$. □

Again, as a consequence of fixing $(0, 0)$ the dual will be generated by it.

Corollary 3. *Let the setup be as in Proposition 4. Then $\ker(\hat{\phi}) = \langle (0, 0) \rangle$.*

Proof. Since $(0, 0) \in E$ has order 3 and is not in $\ker(\phi)$, it follows from $\hat{\phi} \circ \phi = [3]$ that $\phi((0, 0)) \neq \mathcal{O}_{\tilde{E}}$, while $(\hat{\phi} \circ \phi)((0, 0)) = \mathcal{O}_E$. Hence $\phi((0, 0)) \in \ker(\hat{\phi})$, and since $\deg(\hat{\phi}) = 3$ we have that $\ker(\hat{\phi}) = \langle \phi((0, 0)) \rangle$. The result is now immediate by observing that $\phi((0, 0)) = (0, 0)$. □

5.3 Application to Isogeny-Based Cryptography

By doing an analogous analysis as in Sect. 4.3 it is straightforward to see that it is theoretically possible to use the triangular form as above in isogeny-based systems. More specifically, by choosing a basis $E(\mathbb{F}_{p^2})[3^e] = \langle P, Q \rangle$ such that $[3^{e-1}]Q = (0, 0)$ and by only allowing secret kernels of the form $\langle P + [\alpha]Q \rangle$, we can always apply the isogeny from Proposition 4. However, to be seriously considered for implementations the efficiency must be at least on par with those coming from the Montgomery form. Although the computation of A can be done with only two multiplications, we have not been able to reduce the cost of the 3-isogeny evaluation far enough to be considered as efficient as its Montgomery counterpart. Moreover, the x -only tripling formulas (which can for example be obtained by using the 3-isogenies from [BCKL15, Theorem 5.4]) are significantly slower.

Acknowledgements. I would like to thank Craig Costello for valuable suggestions and feedback during the creation of this document, and Chloe Martindale for comments on a first version of the paper, in particular to improve the proof of Theorem 1. I thank the anonymous reviewers of PQCrypto 2018 for their constructive comments.





References

- [Acc99] Accredited Standards Committee X9. American National Standard X9.62-1999, Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA). Technical report, ANSI (1999)
- [BCKL15] Bernstein, D.J., Chuengsatiansup, C., Kohel, D., Lange, T.: Twisted hessian curves. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 269–294. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_15
- [BDL+12] Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High-speed high-security signatures. *J. Cryptogr. Eng.* **2**(2), 77–89 (2012)
- [Ber06] Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_14
- [Brö09] Bröker, R.: Constructing supersingular elliptic curves. *J. Comb. Number Theory* **1**(3), 269–273 (2009)
- [CH17] Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. *Cryptology ePrint Archive, Report 2017/504* (2017)
- [CJL+17] Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient compression of SIDH public keys. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 679–706. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_24
- [CLN16a] Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_21
- [CLN16b] Costello, C., Longa, P., Naehrig, M.: SIDH Library (2016). <http://research.microsoft.com/en-us/downloads/bd5fd4cd-61b6-458a-bd94-b1f406a3f33f/>

- [Cou06] Couveignes, J.M.: Hard Homogeneous Spaces. IACR Cryptology ePrint Archive (2006)
- [DH76] Diffie, W., Hellman, M.E.: New directions in cryptography. *IEEE Trans. Inf. Theory* **22**(6), 644–654 (1976)
- [FJP14] De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.* **8**(3), 209–247 (2014)
- [Gal12] Galbraith, S.D.: *Mathematics of Public Key Cryptography*. Cambridge University Press, Cambridge (2012)
- [Hus04] Husemöller, D.: *Elliptic Curves*. Graduate Texts in Mathematics. Springer, New York (2004). <https://doi.org/10.1007/978-1-4757-5119-2>
- [JF11] Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) *PQCrypto 2011*. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
- [KAK16] Koziel, B., Azarderakhsh, R., Mozaffari-Kermani, M.: Fast hardware architectures for supersingular isogeny diffie-hellman key exchange on FPGA. In: Dunkelmann, O., Sanadhya, S.K. (eds.) *INDOCRYPT 2016*. LNCS, vol. 10095, pp. 191–206. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49890-4_11
- [Kob87] Koblitz, N.: Elliptic curve cryptosystems. *Math. Comput.* **48**, 203–209 (1987)
- [Len87] Lenstra, H.W.: Factoring integers with elliptic curves. *Ann. Math.* **126**, 649–673 (1987)
- [Mil86] 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
- [Mon87] Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. *Math. Comput.* **48**(177), 243–264 (1987)
- [MS16] Moody, D., Shumow, D.: Analogues of Vélú’s formulas for isogenies on alternate models of elliptic curves. *Math. Comput.* **85**(300), 1929–1951 (2016)
- [RS06] Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. IACR Cryptology ePrint Archive, 2006:145 (2006)
- [Sch89] Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) *CRYPTO 1989*. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
- [Sho94] Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: *35th Annual Symposium on Foundations of Computer Science, 1994 Proceedings*, pp. 124–134. IEEE (1994)
- [Sil09] Silverman, J.H.: *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics, 2nd edn. Springer, New York (2009). <https://doi.org/10.1007/978-0-387-09494-6>
- [Vél71] Vélú, J.: Isogénies entre courbes elliptiques. *Comptes Rendus de l’Académie des Sciences des Paris* **273**, 238–241 (1971)
- [ZJP+17] Zanon, G.H.M., Simplicio Jr., M.A., Pereira, G.C.C.F., Doliskani, J., Barreto, P.S.L.M.: Faster isogeny-based compressed key agreement. *Cryptology ePrint Archive*, Report 2017/1143 (2017). <https://eprint.iacr.org/2017/1143>



Faster Isogeny-Based Compressed Key Agreement

Gustavo H. M. Zanon¹, Marcos A. Simplicio Jr¹ ,
Geovandro C. C. F. Pereira² , Javad Doliskani² ,
and Paulo S. L. M. Barreto³ 

¹ Escola Politécnica, University of São Paulo, São Paulo, Brazil
{gzanon,msimplicio}@larc.usp.br

² Institute for Quantum Computing, University of Waterloo, Waterloo, Canada
{geovandro.pereira,javad.doliskani}@uwaterloo.ca

³ University of Washington Tacoma, Tacoma, USA
pbarreto@uw.edu

Abstract. Supersingular isogeny-based cryptography is one of the more recent families of post-quantum proposals. An interesting feature is the comparatively low bandwidth occupation in key agreement protocols, which stems from the possibility of key compression. However, compression and decompression introduce a significant overhead to the overall processing cost despite recent progress. In this paper we address the main processing bottlenecks involved in key compression and decompression, and suggest substantial improvements for each of them. Some of our techniques may have an independent interest for other, more conventional areas of elliptic curve cryptography as well.

1 Introduction

In the Supersingular Isogeny Diffie-Hellman (SIDH) protocol [9], the two parties need to exchange a representation of their public keys each consisting of an elliptic curve E together with two points P, Q on E . The curve E is supersingular and is defined over an extension field \mathbb{F}_{p^2} for a prime of the form $p = \ell_A^m \ell_B^n - 1$ where ℓ_A, ℓ_B are small primes, usually equal to 2 and 3, respectively. Originally, this exchange was done using triples of the form (E, x_P, x_Q) where $E : y^2 = x^3 + ax + b$ and x_P, x_Q are the abscissas of P and Q . Two extra bits were also needed to recover the correct y -coordinates. Thus, the public keys are transferred using essentially the four elements $a, b, x_P, x_Q \in \mathbb{F}_{p^2}$ which require $8 \log p$ bits.

A different representation of the SIDH public keys was proposed by [1] that reduced the size to $4 \log p$ bits. The idea was to first represent the curve E using its j -invariant, which is an element of \mathbb{F}_{p^2} , rather than the coefficients a, b . This way E is represented using $2 \log p$ bits. The isomorphism class of an elliptic curve is uniquely determined by its j -invariant. Second, since the points P, Q are always in the torsion subgroups $E[\ell_A^m]$ or $E[\ell_B^n]$, they can be represented using elements of $\mathbb{Z}_t \oplus \mathbb{Z}_t$ where either $t = \ell_A^m$ or $t = \ell_B^n$. Since the parameters

are such that $\ell_A^m \approx \ell_B^n$, a pair $(t_1, t_2) \in \mathbb{Z}_t \oplus \mathbb{Z}_t$ is represented using $2 \log p$ bits. This reduction of size of the public keys, however, comes with a rather high computational overhead. The conversion between the coefficients a, b of a curve E and its j -invariant is done efficiently; the expensive part is the conversion between elements of $\mathbb{Z}_t \oplus \mathbb{Z}_t$ and the points P, Q . As reported in [1], the compression phase for each party was slower than a full round of uncompressed key exchange by a factor of more than 10 times.

Costello *et al.* [5] further improved the key compression scheme by reducing the public key sizes to $3.5 \log p$ bits and decreasing the runtime by almost an order of magnitude. With this improvement, the key compression phase for each party is as fast as one full round of uncompressed key exchange. This certainly motivates the idea of including the compression and decompression phases as default parts of SIDH. However, compared to the currently deployed (classical) schemes, the compression/decompression runtime is unfavorable, requiring further research on speedup techniques.

Our contributions. We propose new algorithms that further decrease the runtime of SIDH compression and decompression. In contrast to previous works that have deployed “known” algorithms to optimize the performance of key compression, some of the algorithms presented here are new and of broader interest than isogeny-based crypto. A summary of the improvements follows.

- Constructing torsion bases. Assuming the usual parameters $\ell_A = 2, \ell_B = 3$, we improve basis generation for both $E[2^m]$ and $E[3^n]$. To generate a basis for the 2^m -torsion, we propose an algorithm dubbed *entangled basis* generation. This algorithm is around $15.9\times$ faster than the usual basis generation presented in [5]. For the 3^n -torsion, we observed that the naive algorithm is more efficient (both in theory and practice) than the explicit 3-descent of [12] used by Costello *et al.* [5].
- Computing discrete logarithms. Inspired by the *optimal strategy* method of [6] to compute smooth degree isogenies, we propose an algorithm to compute discrete logs in the group μ_{ℓ^n} where ℓ is a small prime. For a window of size $w = 6$, our algorithm is $3.9\times$ and $4.6\times$ faster than the algorithm used by [5] for the groups $\mu_{2^{372}}$ and $\mu_{3^{239}}$ respectively.
- Pairing computation. We exploit the special shapes of pairs of points generated as entangled bases and the existence of a subfield dismissed by [5] to optimize the Tate pairing. We achieve a speedup of $1.4\times$ for the pairing phase over the algorithms used by [5] for both binary and ternary pairings.
- Other improvements. We introduce *reverse basis decomposition*, which combined with the previous improvements, allows for further optimizations of compression and decompression. For example each party only needs to compute 4 pairings rather than 5. Also, two expensive cofactor multiplications by 3^n are saved during Bob’s compression, and one cofactor multiplication by 3^n is saved during Alice’s decompression.

We have implemented the above improvements on top of (the then-latest version of) the Microsoft SIDH library [10]. The library is designed for the specific prime

$p = 2^{372}3^{239} - 1$ of size $\log p = 751$ bits. Our software can be found at <https://github.com/geovandro/PQCrypto-SIDH>.

1.1 Notations and Conventions

For simplicity, we assume that finite field arithmetic is carried out in a base field \mathbb{F}_p and its quadratic extension \mathbb{F}_{p^2} for a prime p of form $p := 2^m \cdot 3^n - 1$ for some $m > 2$ and $n > 1$, so that $p \equiv 3 \pmod{4}$. The quadratic extension $\mathbb{F}_{p^2}/\mathbb{F}_p$ is represented as $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/\langle i^2 + 1 \rangle$, and arithmetic closely mimics that of the complex numbers.

All curves are represented using the Montgomery model unless otherwise specified. We follow the convention of using subscripts A and B for Alice and Bob, respectively. For example, the secret isogeny ϕ_A is computed by Alice and her public parameters are denoted by the points P_A, Q_A and the curve E_A . Similarly, Bob's isogeny is denoted by ϕ_B , and his public parameters are P_B, Q_B, E_B .

We denote by $\mathbf{i}, \mathbf{c}, \mathbf{m}, \mathbf{s}$, and \mathbf{a} the costs of inverting, cubing, multiplying, squaring, and adding/subtracting/shifting in \mathbb{F}_p , respectively, and by $\mathbf{I}, \mathbf{C}, \mathbf{M}, \mathbf{S}$, and \mathbf{A} the costs of the corresponding operations in \mathbb{F}_{p^2} . We disregard the cost of changing a sign (for instance, when handling the conjugate of a field element). The costs of the \mathbb{F}_{p^2} operations relative to the costs of operations in \mathbb{F}_p can be approximated by $1\mathbf{I} = 1\mathbf{i} + 2\mathbf{m} + 2\mathbf{s} + 1\mathbf{a}$, $1\mathbf{C} = 2\mathbf{m} + 2\mathbf{s} + 6\mathbf{a}$, $1\mathbf{M} = 3\mathbf{m} + 5\mathbf{a}$, $1\mathbf{S} = 2\mathbf{m} + 3\mathbf{a}$, and $1\mathbf{A} = 2\mathbf{a}$, by using the finite-field analogues to well-known Viète multiple-angle trigonometric identities [15, Formulas 5.68 and 5.69].

2 Reverse Basis Decomposition

In this section, we use *reverse basis decomposition* to speed up both Alice's and Bob's key compression by saving one pairing computation. Later in Sect. 3.1 we show that, when combined with an entangled basis generation, this technique will allow for avoiding two cofactor multiplications by 3^n in Bob's key compression and one in Alice's key decompression. We prove our results from Alice's point of view. The proofs for Bob are similar.

The main previous idea to achieve key compression [1, 5] is the following: instead of transmitting points $\phi_A(P_B), \phi_A(Q_B) \in E_A[3^n]$, which are represented by two abscissas in \mathbb{F}_{p^2} and consume $4 \log p$ bits, Alice computes a canonical basis $R_1, R_2 \in E_A[3^n]$ and expresses the expanded public key in that basis as $\phi_A(P_B) = a_0 R_1 + b_0 R_2$ and $\phi_A(Q_B) = a_1 R_1 + b_1 R_2$. In matrix notation,

$$\begin{bmatrix} \phi_A(P_B) \\ \phi_A(Q_B) \end{bmatrix} = \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}. \quad (1)$$

This representation consists of four smaller integers $(a_0, b_0, a_1, b_1) \in (\mathbb{Z}/3^n\mathbb{Z})^4$ of total size $2 \log p$ bits as suggested in [1]. This was improved in [5] by transmitting only the triple $(a_0^{-1}b_0, a_0^{-1}a_1, a_0^{-1}b_1) \in (\mathbb{Z}/3^n\mathbb{Z})^3$ or $(b_0^{-1}a_0, b_0^{-1}a_1, b_0^{-1}b_1) \in (\mathbb{Z}/3^n\mathbb{Z})^3$ depending on whether a_0 or b_0 is invertible. Therefore, only $(3/2) \log p$,

plus one bit indicating the invertibility of a_0 or b_0 modulo 3^n , is needed. In the above mentioned techniques, the coefficients a_0, b_0, a_1, b_1 can be computed using five Tate pairings given by

$$\begin{aligned}
 g_0 &= e_{3^n}(R_1, R_2) \\
 g_1 &= e_{3^n}(R_1, \phi_A(P_B)) = e_{3^n}(R_1, a_0R_1 + b_0R_2) = g_0^{b_0} \\
 g_2 &= e_{3^n}(R_1, \phi_A(Q_B)) = e_{3^n}(R_1, a_1R_1 + b_1R_2) = g_0^{b_1} \\
 g_3 &= e_{3^n}(R_2, \phi_A(P_B)) = e_{3^n}(R_2, a_0R_1 + b_0R_2) = g_0^{-a_0} \\
 g_4 &= e_{3^n}(R_2, \phi_A(Q_B)) = e_{3^n}(R_2, a_1R_1 + b_1R_2) = g_0^{-a_1}.
 \end{aligned} \tag{2}$$

From this, Alice can recover $a_0, b_0, a_1,$ and b_1 by solving discrete logs in a multiplicative subgroup of smooth order 3^n using the Pohlig-Hellman algorithm.

Now since $\phi_A(P_B)$ and $\phi_A(Q_B)$ also form a basis of $E_A[3^n]$, we see that the coefficient matrix in (1) is invertible modulo 3^n . So, we can write

$$\begin{bmatrix} R_1 \\ R_2 \end{bmatrix} = \begin{bmatrix} c_0 & d_0 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} \phi_A(P_B) \\ \phi_A(Q_B) \end{bmatrix} \tag{3}$$

by inverting the matrix in (1). Changing the roles of the bases $\{R_1, R_2\}$ and $\{\phi_A(P_B), \phi_A(Q_B)\}$ in (2) we get

$$\begin{aligned}
 h_0 &= e_{3^n}(\phi_A(P_B), \phi_A(Q_B)) \\
 h_1 &= e_{3^n}(\phi_A(P_B), R_1) = e_{3^n}(\phi_A(P_B), c_0\phi_A(P_B) + d_0\phi_A(Q_B)) = h_0^{d_0} \\
 h_2 &= e_{3^n}(\phi_A(P_B), R_2) = e_{3^n}(\phi_A(P_B), c_1\phi_A(P_B) + d_1\phi_A(Q_B)) = h_0^{d_1} \\
 h_3 &= e_{3^n}(\phi_A(Q_B), R_1) = e_{3^n}(\phi_A(Q_B), c_0\phi_A(P_B) + d_0\phi_A(Q_B)) = h_0^{-c_0} \\
 h_4 &= e_{3^n}(\phi_A(Q_B), R_2) = e_{3^n}(\phi_A(Q_B), c_1\phi_A(P_B) + d_1\phi_A(Q_B)) = h_0^{-c_1}.
 \end{aligned} \tag{4}$$

The first pairing in (4) is computed as $h_0 = e_{3^n}(P_B, \hat{\phi}_A \circ \phi_A(Q_B)) = e_{3^n}(P_B, [\deg \phi_A]Q_B) = e_{3^n}(P_B, Q_B)^{2^m}$, which only depends on the public parameters P_B, Q_B and m . Therefore, it can be computed once and for all and be included in the public parameters. In particular, only the pairings h_1, h_2, h_3 and h_4 need to be computed at runtime. The discrete logs are computed as before using Pohlig-Hellman, yielding $c_0 = -\log_{h_0} h_3, d_0 = \log_{h_0} h_1, c_1 = -\log_{h_0} h_4$ and $d_1 = \log_{h_0} h_2$. Next, Alice inverts the computed coefficients matrix of (3) to obtain the coefficient matrix of (1). Explicitly,

$$\begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \end{bmatrix} = \frac{1}{D} \begin{bmatrix} d_1 & -d_0 \\ -c_1 & c_0 \end{bmatrix}$$

where $D = c_0d_1 - c_1d_0$. Notice that the extra inversion of D^{-1} does not need to be carried out when using the technique in [5]. More precisely, since at least one of d_0 and d_1 , say d_1 , is invertible modulo 3^n , Alice transmits the tuple

$$\begin{aligned}
 (a_0^{-1}b_0, a_0^{-1}a_1, a_0^{-1}b_1) &= (-d_1^{-1}DD^{-1}d_0, -d_1^{-1}DD^{-1}c_1, d_1^{-1}DD^{-1}c_0) \\
 &= (-d_1^{-1}d_0, -d_1^{-1}c_1, d_1^{-1}c_0)
 \end{aligned}$$

which is independent of D .

3 Entangled Basis Generation

We now introduce a technique to create a complete basis of the 2^m -torsion from a single (albeit specific) point of order 2^m . In other words, the cost involved is essentially that of creating a generator for a single subgroup of order 2^m in $E[2^m]$: a generator for the linearly independent subgroup becomes immediately available almost for free. Consequently, the linear independence test consisting of two scalar multiplications by 2^{m-1} can be avoided. This is akin to distortion maps even though none is typically available for the curves involved in SIDH. We call the resulting bases “entangled” by analogy with the quantum phenomenon whereby the properties of one entity are entirely determined by the properties of another entity despite their separation⁴.

In order to construct an entangled basis $\langle P, Q \rangle = E[2^m]$ for $E : y^2 = x^3 + Ax^2 + x$, we somewhat “subvert” the original Elligator 2 formulas [3] under a different motivation than encoding points to random strings: obtaining two linearly independent points on E at once. Herein the value $t := ur^2$, for $u \in \mathbb{F}_{p^2}$ and $r \in \mathbb{F}_p^*$, will be a square rather than a non-square. The new construction is proved in Theorem 1.

Theorem 1. *Given a Montgomery supersingular elliptic curve $E_A/\mathbb{F}_{p^2} : y^2 = x(x^2 + Ax + 1)$ where $p = 2^m \cdot 3^n - 1$, $\#E_A(\mathbb{F}_{p^2}) = (p + 1)^2$, and $A \neq 0$, let $t \in \mathbb{F}_{p^2}$ be a field element such that $t^2 \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$, and let $x_1 := -A/(1 + t^2)$ be a quadratic non-residue that defines the abscissa of a point $P_1 \in E_A(\mathbb{F}_{p^2})$. Then $x_2 := -x_1 - A$ defines the abscissa of another point $P_2 \in E_A(\mathbb{F}_{p^2})$ such that $\langle [h]P_1, [h]P_2 \rangle = E_A[2^m]$, where $h := 3^n$ is the cofactor of the 2^m -torsion group.*

Proof. Since $x_2 = t^2x_1$, both abscissas are quadratic non-residues and by [8, Chap. 1 (Sect. 4), Theorem 4.1] the two points $P_1 = (x_1, y_1)$, $P_2 = (t^2x_1, ty_1)$, with $x_1 + t^2x_1 + A = 0$, are not in $[2]E_A$. So the points $[h]P_1$ and $[h]P_2$ are full 2^m -torsion points. To prove that $h \cdot P_1, h \cdot P_2$ generate $E_A[2^m]$ we have to prove that $[h \cdot 2^{m-1}](P_1 - P_2) \neq 0$, or equivalently that $(u, v) = P_1 + (-P_2) \notin [2]E_A$.

By the addition law [14, Algorithm 2.3] on E_A we get

$$\begin{aligned} \lambda &= \frac{y_2 - y_1}{x_2 - x_1} = \frac{-ty_1 - y_1}{t^2x_1 - x_1} = \frac{-(t + 1)y_1}{(t^2 - 1)x_1} = \frac{-y_1}{(t - 1)x_1}, \\ \mu &= \frac{y_1x_2 - y_2x_1}{x_2 - x_1} = \frac{t(t + 1)y_1x_1}{(t + 1)(t - 1)x_1} = \frac{y_1}{(t - 1)x_1}tx_1 = -\lambda tx_1, \\ u &= \lambda^2 - A - x_1 - x_2 = \lambda^2, \\ v &= -\lambda u - \mu = -\lambda u - (-\lambda tx_1) = -\lambda(u - tx_1). \end{aligned}$$

From the above equalities we see that $v^2 = \lambda^2(u - tx_1)^2 = u(u^2 + Au + 1)$ and hence $u^2 + Au + 1 = (u - tx_1)^2$. Let $w := u - tx_1 = \sqrt{u^2 + Au + 1}$. Then $1 - (u - w)^2 = 1 - t^2x_1^2 = x_1^2 + Ax_1 + 1$, which is a quadratic non-residue

⁴ We stress, however, that here the naming is purely analogous: there is no quantum process involved in the construction.

because x_1 is itself a quadratic non-residue while their product is obviously a square, $x_1(x_1^2 + Ax_1 + 1) = y_1^2$. A straightforward calculation shows that $(1 - (u + w)^2)(1 - (u - w)^2) = u^2(A^2 - 4)$. But $A^2 - 4$ is a quadratic residue since E_A has the full 2-torsion over \mathbb{F}_{p^2} . Therefore, both $(u \pm w)^2 - 1$ have the same quadratic residuosity, that is, they are both quadratic non-residues by the above.

Now⁵ assume by contradiction that $P_1 - P_2 \in [2]E_A$, i.e. there is a point $(x, y) \in E_A(\mathbb{F}_{p^2})$ such that $[2](x, y) = (u, v)$. From the doubling formula on E_A we get

$$u = \frac{(x^2 - 1)^2}{4x(x^2 + Ax + 1)}.$$

From this we get a quartic equation $(x^2 - 1)^2 - 4ux(x^2 + Ax + 1) = 0$. Since $x \neq 0$, we can divide both sides by x^2 and rearrange some terms to get

$$\left(x + \frac{1}{x}\right)^2 - 4u\left(x + \frac{1}{x}\right) - 4(Au + 1) = 0.$$

From this we obtain

$$x + \frac{1}{x} = \frac{4u \pm \sqrt{16(u^2 + Au + 1)}}{2} = \frac{4u \pm 4w}{2} = 2(u \pm w).$$

In turn, from this we get $x^2 - 2(u \pm w)x + 1 = 0$. Again since $x \in \mathbb{F}_{p^2}$, the discriminant $4(u \pm w)^2 - 4$, and hence at least one of the $(u \pm w)^2 - 1$ must be a quadratic residue. But this contradicts the earlier observation that $(u \pm w)^2 - 1$ are both quadratic non-residues. Therefore $P_1 - P_2 \notin [2]E$, yielding the claim that $\langle [h]P_1, [h]P_2 \rangle = E_A[2^m]$. \square

In practice, one can efficiently implement entangled basis generation as follows. Let $u_0 \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ such that $u := u_0^2 \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$, e.g. $u_0 = 1 + i$ and $u = 2i$. Define two separate tables of pairs (r, v) with $v := 1/(1 + ur^2)$:

- table T_1 contains pairs (r, v) in which v is quadratic non-residue,
- table T_2 contains pairs (r, v) in which v is quadratic residue.

Performing one quadraticity test on A , only once per curve, and restricting table lookup to the table of opposite quadraticity ensures that $x := -Av$ is a non-square. Repeating quadraticity tests to ensure that a corresponding y exists, and completing one square root extraction in \mathbb{F}_{p^2} to obtain y , one gets 2 points whose orders are multiples of 2^m at once. This is detailed in Algorithm 3.1.

Let us compare the number of operations required by the entangled basis algorithm with the plain basis generation algorithm used in Costello *et al.* [5].

Entangled basis: Testing the quadraticity of A takes $(m + n + 1)\mathbf{s} + n\mathbf{m}$. The main loop runs twice on average at a cost $2(m + n + 1)\mathbf{s} + (2n + 22)\mathbf{m}$. The

⁵ This part closely follows the idea behind [8, Chap. 1 (Sect. 4), Theorem 4.1].

last stage is to complete a square root and costs $(m + n - 1)\mathbf{s} + (n + 1)\mathbf{m} + 1\mathbf{i}$. The total cost of the algorithm is then

$$(4m + 4n + 2)\mathbf{s} + (4n + 23)\mathbf{m} + 1\mathbf{i}.$$

Plain basis: To get the abscissa of a point on the curve takes $(2n + 22)\mathbf{m} + 2(m + n + 1)\mathbf{s}$. Clearing the cofactor 3^n requires n point triplings at a cost $32n\mathbf{m}$. We also need to compute $m - 1$ point doublings for linear independence check that is required in the next steps. So obtaining the first basis point costs $(34n + 16m + 6)\mathbf{m} + 2(m + n + 1)\mathbf{s}$. The second basis point is obtained exactly the same way, except that we also need a linear independence check. This is done in a loop that runs twice on average. The expected cost of obtaining the second point is then twice the cost of obtaining the first point including the $m - 1$ doublings step. The last stage of the algorithm is to recover the y coordinates of the points which costs $(4m + 4n)\mathbf{s} + (4n + 36)\mathbf{m} + 2\mathbf{i}$. Adding all these, the total cost of the algorithm is

$$(10m + 10n + 6)\mathbf{s} + (48m + 106n + 54)\mathbf{m} + 2\mathbf{i}.$$

For the values $m = 372$ and $n = 239$, and assuming $\mathbf{s} = 0.8\mathbf{m}$ and $\mathbf{i} = 100\mathbf{m}$, we get the performance ratio of 15.92.

Algorithm 3.1. Entangled basis generation for $E[2^m](\mathbb{F}_{p^2}) : y^2 = x^3 + Ax^2 + x$

INPUT: $A = a + bi \in \mathbb{F}_{p^2}$; $u_0 \in \mathbb{F}_{p^2} : u := u_0^2 \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$; tables T_1, T_2 of pairs $(r \in \mathbb{F}_p, v = 1/(1 + ur^2) \in \mathbb{F}_{p^2})$ of QNR and QR.

OUTPUT: $\{S_1, S_2\}$ such that $\langle [3^n]S_1, [3^n]S_2 \rangle = E[2^m](\mathbb{F}_{p^2})$.

- 1: $z \leftarrow a^2 + b^2, \quad s \leftarrow z^{(p+1)/4}$
 - 2: $T \leftarrow (s^2 \stackrel{?}{=} z) T_1 : T_2 \quad //$ select proper table by testing quadraticity of A
 - 3: **repeat**
 - 4: lookup next entry (r, v) from T
 - 5: $x \leftarrow -A \cdot v \quad //$ NB: x nonsquare
 - 6: $t \leftarrow x \cdot (x^2 + A \cdot x + 1) \quad //$ test quadraticity of $t = c + di$
 - 7: $z \leftarrow c^2 + d^2, \quad s \leftarrow z^{(p+1)/4}$
 - 8: **until** $s^2 = z \quad //$ compute $y \leftarrow \sqrt{x^3 + A \cdot x^2 + x}$
 - 9: $z \leftarrow (c + s)/2, \quad \alpha \leftarrow z^{(p+1)/4}, \quad \beta \leftarrow d \cdot (2\alpha)^{-1}$
 - 10: $y \leftarrow (\alpha^2 \stackrel{?}{=} z) \alpha + \beta i : -\beta + \alpha i \quad //$ compute basis
 - 11: **return** $S_1 \leftarrow (x, y), S_2 \leftarrow (ur^2x, u_0ry)$
-

For the sake of completeness, we remark that the entangled basis method requires the Montgomery curve coefficient A to be nonzero, and hence does not apply to the curve $E_0 : y^2 = x^3 + x$. That case is not relevant for compression, but it can be handled through the use of the usual distortion map $\tau : E_0 \rightarrow E_0$ defined by $\tau(x, y) = (-x, iy)$. However, as we discuss in Sect. 7, both basis points must be $E_0(\mathbb{F}_{p^2}) \setminus E_0(\mathbb{F}_p)$ in this case as well, and since the distortion map by itself does not guarantee this property, it has to be properly checked.

3.1 Avoiding Cofactor Multiplication

Combining reverse basis decomposition and entangled basis generation enables us to further avoid two scalar multiplications by the large cofactor 3^n during Bob’s public key compression, and one during Alice’s decompression. First notice that Algorithm 3.1 already incorporates the mentioned optimization, i.e. the output points S_1 and S_2 satisfy $(R_1, R_2) := ([3^n]S_1, [3^n]S_2)$ such that $\langle R_1, R_2 \rangle = E[2^m]$. This is only possible because in reverse basis decomposition the Tate pairings h_i take the points S_i in their second argument which does not need to be necessarily cofactor-reduced. In this case, for $R_1 = c_0\phi_B(P_A) + d_0\phi_B(Q_A)$ and $R_2 = c_1\phi_B(P_A) + d_1\phi_B(Q_A)$, the respective pairing computations are

$$\begin{aligned} k_0 &= e_{2^m}(\phi_B(P_A), \phi_B(Q_A)) \\ k_1 &= e_{2^m}(\phi_B(P_A), S_1) = e_{2^m}(\phi_B(P_A), [3^{-n}]R_1) = k_0^{3^{-n}d_0} \\ k_2 &= e_{2^m}(\phi_B(P_A), S_2) = e_{2^m}(\phi_B(P_A), [3^{-n}]R_2) = k_0^{3^{-n}d_1} \\ k_3 &= e_{2^m}(\phi_B(Q_A), S_1) = e_{2^m}(\phi_B(Q_A), [3^{-n}]R_1) = k_0^{-3^{-n}c_0} \\ k_4 &= e_{2^m}(\phi_B(Q_A), S_2) = e_{2^m}(\phi_B(Q_A), [3^{-n}]R_2) = h_0^{-3^{-n}c_1}. \end{aligned}$$

Thus, the discrete logarithms are the desired ones up to a factor 3^{-n} , and given by $\hat{c}_0 = -\log_{k_0} k_3 = 3^{-n}c_0$, $\hat{d}_0 = \log_{k_0} k_1 = 3^{-n}d_0$, $\hat{c}_1 = -\log_{k_0} k_4 = 3^{-n}c_1$, and $\hat{d}_1 = \log_{k_0} k_2 = 3^{-n}d_1$. Notice that $3^{-n} \bmod 2^m$ must be odd which implies that \hat{c}_0 or \hat{d}_0 is invertible if and only if c_0 or d_0 is invertible. Similar to the situation in Sect. 2, when using the compression with only 3 coefficients as in [5] Bob transmits exactly the original coefficients: assuming \hat{c}_0 is invertible, then

$$\begin{aligned} (\hat{c}_0^{-1}\hat{d}_0, \hat{c}_0^{-1}\hat{c}_1, \hat{c}_0^{-1}\hat{d}_1) &= (c_0^{-1}3^n3^{-n}d_0, c_0^{-1}3^n3^{-n}c_1, c_0^{-1}3^n3^{-n}d_1) \\ &= (c_0^{-1}d_0, c_0^{-1}c_1, c_0^{-1}d_1) \end{aligned}$$

The derivation when d_0 is invertible is analogous.

To decompress Bob’s public key, Alice needs to perform a single cofactor multiplication by 3^n as follows. Assume that a_0 is invertible modulo 2^m so that Alice receives the triple $(a_0^{-1}b_0, a_0^{-1}a_1, a_0^{-1}b_1)$. She needs to compute the kernel $\ker(\phi_{AB}) = \langle \phi_B(P_A) + sk_A \cdot \phi_B(Q_A) \rangle$ which can be written as

$$\langle a_0R_1 + b_0R_2 + sk_A \cdot (a_1R_1 + b_1R_2) \rangle = \langle (a_0 + sk_Aa_1)R_1 + (b_0 + sk_Ab_1)R_2 \rangle$$

As noted in [5], one computes $\ker(\phi_{AB})$ as $a_0^{-1} \ker(\phi_{AB}) = \langle (1 + sk_Aa_0^{-1}a_1)R_1 + (a_0^{-1}b_0 + sk_Aa_0^{-1}b_1)R_2 \rangle$, which can be done with one scalar multiplication and one point addition by writing $\ker(\phi_{AB}) = \langle R_1 + (1 + sk_Aa_0^{-1}a_1)^{-1}(a_0^{-1}b_0 + sk_Aa_0^{-1}b_1)R_2 \rangle$. Now if Alice uses Algorithm 3.1, she obtains an entangled basis $\{S_1, S_2\}$ such that $(R_1, R_2) = ([3^n]S_1, [3^n]S_2)$. She can then compute $T = \langle S_1 + (1 + sk_Aa_0^{-1}a_1)^{-1}(a_0^{-1}b_0 + sk_Aa_0^{-1}b_1)S_2 \rangle$ first and then recover the correct kernel $\ker(\phi_{AB}) = \langle [3^n]T \rangle$ by performing one cofactor scalar multiplication.

3.2 On Basis Generation for $E[3^n]$

The entangled basis approach does not immediately generalize to the ternary case. As a consequence, to generate bases for $E[3^n]$ we adopted the naïve approach of randomly picking candidate points and testing them for the correct order and linear independence.

Costello *et al.* suggest the use of a 3-descent approach based on a result by Schaefer and Stoll [12], and claim significant performance gains. However, we were unable to reproduce and thus verify their claims. On the contrary, the naïve method is observed to be always faster than 3-descent, with a cost ratio $\mathcal{C}_{\text{naïve}}/\mathcal{C}_{3\text{-descent}} \approx 0.87$ that runs against their claim. A detailed analysis appears to corroborate the observed cost ratio above. We defer further discussion for the extended version of this paper [16].

We briefly note that, while the naïve approach incurs a substantial cost that seems unavoidable at key compression, the knowledge gained in the process (in the form of the actual counters that specify the points in the Elligator construction) could be then shared between Alice and Bob, speeding up the latter's work at key decompression. For a very modest increase in Alice's key size (for instance, a single extra byte for each of the two basis points would provide space that is only exceeded with probability well below 2^{-400}), Bob's $E[3^n]$ basis generation would get about 31% faster, and his full decompression of Alice's key would become about 22% faster. We also defer a detailed discussion of this technique for the extended version of this paper [16].

4 Pairing Computation

The pairing computation techniques by Costello *et al.* [5] are based on curves in a variant of the Montgomery model, with projective coordinates (X^2, XZ, Z^2, YZ) , which turned out to be the best setting among several models they assessed. We will argue that the older and today less favored short Weierstraß model leads to more efficient pairing algorithms. For convenience, we extend Jacobian coordinates $[X : Y : Z]$ with a fourth component, $[X : Y : Z : T]$ with $T = Z^2$.

Interestingly, Costello *et al.* dismiss the technique of denominator elimination [2] and keep numerators and denominators separate during pairing evaluation. We point out, however, that pairing values are defined over \mathbb{F}_{p^2} and the inverse of a field element $a+bi$ is $(a-bi)/(a^2+b^2)$. Hence, rather than keeping a separate denominator $a+bi$ one can simply and immediately multiply the pairing value by the conjugate $a-bi$ instead; the result only differs from the original one by a denominator consisting of the norm $a^2+b^2 \in \mathbb{F}_p$, and this denominator does get eliminated by the final exponentiation in the reduced Tate pairing computation. This leaves the cost of pairing computation unchanged, but it simplifies the implementation as it entirely does away with separate numerators and denominators.

Let $r \geq 0$ be the pairing order. For embedding degree $k = 2$, $r \mid \Phi_2(p) = p + 1 = 2^m \cdot 3^n$, and by construction r is always either 2^m or 3^n . We will be

interested in computing reduced Tate pairings of order r , whose first argument must have that order as well. In the case of compressed SIDH keys, pairings of the following forms are computed together (recall that a fifth pairing $e_0 := e_r(P, Q) = e_r(P_0, Q_0)^{\deg \phi}$ is readily available through precomputation):

$$e_1 := e_r(P, R_1), e_2 := e_r(P, R_2), e_3 := e_r(Q, R_1), e_4 := e_r(Q, R_2),$$

where the first two pairings share the same first argument P , and next two pairings share the same first argument Q .

From now on, we will split the discussion into two cases: binary-order pairings, $r = 2^m$, and ternary-order pairings, $r = 3^n$.

4.1 Binary-Order Pairings

The computation of the reduced Tate pairing $e_r(P, Q)$ of order $r = 2^m$ proceeds as described in Algorithm 4.1, which requires doubling a point $V \in E(\mathbb{F}_{p^2})$. The doubling formulas in Jacobian coordinates have a single exception, that occurs when the point being doubled has order 2. That is, when $y = 0$, since the angular coefficient of the tangent to the curve at that point becomes undefined. That exception, however, can only occur deterministically in the scenario contemplated here, namely at the last step of the Miller loop; since by definition the first pairing argument is always a point of order 2^m , chosen by the very entity that is computing the pairing.

Besides, the difference in running time reveals no private information, since the pairing arguments are either already public for being part of a conventional torsion basis, or else are about to be made public for being part of a public key.

Algorithm 4.1. Tate2(P, Q): basic reduced Tate pairing of order $r = 2^m$:

INPUT: points P, Q .

OUTPUT: $e_r(P, Q)$.

- 1: $f \leftarrow 1, V \leftarrow P$
 - 2: **for** $i \leftarrow 0$ **to** $m - 1$ **do**
 - 3: $f \leftarrow f^2 \cdot g_{V,V}(Q)/g_{[2]V}(Q), V \leftarrow [2]V$
 - 4: **end for**
 - 5: **return** $e_r(P, Q) \leftarrow f^{(p^2-1)/r}$
-

The most efficient doubling method known for Jacobian coordinates is due to Bernstein and Lange [4]. Let $V = [X : Y : Z : T]$ and $[2]V = [X' : Y' : Z' : T']$ in the extended coordinate system defined above. Then

$$\begin{array}{lll} X_2 \leftarrow X^2; & Y_2 \leftarrow Y^2; & Y_4 \leftarrow Y_2^2; \\ S \leftarrow 2((X + Y_2)^2 - X_2 - Y_4); & M \leftarrow 3X_2 + A \cdot T^2; & X' \leftarrow M^2 - 2S; \\ Y' \leftarrow M \cdot (S - X') - 8Y_4; & Z' \leftarrow (Y + Z)^2 - Y_2 - T; & T' \leftarrow (Z')^2; \end{array}$$

The cost is $2\mathbf{M} + 8\mathbf{S} + 15\mathbf{A}$. The curve coefficient A typically lacks any structure that might enable optimizations, and hence incurs full multiplication cost.

This algorithm yields several intermediate values that are useful in the calculation of a function equivalent to $g_{V,V}(Q)/g_{[2]V}(Q)$, namely $\tilde{g}_2(V, Q) := [M \cdot (T \cdot x - X) + W - L \cdot y] \cdot R \cdot (T' \cdot x - X')^*$ when $[2]V \neq O$, $\tilde{g}_2(V, Q) := (T \cdot x - X) \cdot T^*$ when $-V = V \neq O$, or simply $\hat{g}_2(V, Q) := 1$ when $V = O$, where $L := Z' \cdot T$, $R := Z' \cdot T^*$, $W := 2Y_2$. Denominators in the base field ($|Z^2 \cdot (T' \cdot x - X')|^2$ in the first case, $|Z^2|^2$ in the others) are eliminated.

One can further optimize the computation of a function $\hat{g}_2(V, Q)$ equivalent to $\tilde{g}_2(V, Q)$. First, the expression $T' \cdot x - X'$ that occurs at one step will play the role of $T \cdot x - X$ at the next step, so one can simply store it from one step to the next and thus save $1\mathbf{M}$. Second, one can show that all R and T^* factors that appear in the definition of $\tilde{g}_2(V, Q)$ are irrelevant to the pairing value, and can be omitted. We defer the details for the extended version of this paper [16].

Consequently, initializing $h \leftarrow T \cdot x - X$ before Miller's loop at a cost of $1\mathbf{M}$ per pairing, the value g of function $\hat{g}_2(V, Q)$ can be evaluated as

$$g \leftarrow M \cdot h + W - L \cdot y; h \leftarrow T' \cdot x - X'; g \leftarrow g \cdot h^*;$$

at a cost of $4\mathbf{M} + 3\mathbf{A}$ per step of Miller's loop, except at the final step, when it is simply $g \leftarrow h$. The cost of computing $L \leftarrow Z' \cdot T$ alone is $1\mathbf{M}$ and that of computing W is $1\mathbf{A}$. This completes the line function construction.

Finally, updating f at each step as $f \leftarrow f^2 \cdot \hat{g}_2(V, Q)$ incurs a cost $1\mathbf{S}$ to compute the complex square f^2 , plus $1\mathbf{M}$ to compute $f^2 \cdot \hat{g}_2(V, Q)$ from f^2 and $\hat{g}_2(V, Q)$. Therefore, the proposed variant has the following overall cost per step, where the shared part is amortized among parallel pairings that share the same first argument:

- (shared) cost of point doubling and line function construction: $2\mathbf{M} + 8\mathbf{S} + 15\mathbf{A} + 1\mathbf{M} + 1\mathbf{A} = 25\mathbf{m} + 71\mathbf{a}$;
- (individual) cost of line function evaluation and accumulation: $4\mathbf{M} + 3\mathbf{A} + 1\mathbf{S} + 1\mathbf{M} = 17\mathbf{m} + 34\mathbf{a}$.

By comparison, the Costello *et al.* [5] technique has the following costs:

- (shared) cost of point doubling and line function construction: $9\mathbf{M} + 5\mathbf{S} + 1\mathbf{s} + 7\mathbf{a} = 37\mathbf{m} + 1\mathbf{s} + 67\mathbf{a}$;
- (individual) cost of line function evaluation and accumulation: $5\mathbf{M} + 2\mathbf{S} + 2\mathbf{s} + 1\mathbf{a} = 19\mathbf{m} + 2\mathbf{s} + 32\mathbf{a}$.

Pairings on an entangled basis. If two pairings $e(P, R_1)$, $e(P, R_2)$ sharing the same first argument P are computed on an entangled basis $R_1 = (x_1, y_1)$, $R_2 = (x_2, y_2)$ with $x_2 = t^2 \cdot x_1$, $y_2 = t \cdot y_1$ with carefully chosen t , one can slightly improve the line function evaluation and accumulation. For $t = (1 + i)r$ and $t^2 = 2ir^2$ with some small $r \in \mathbb{F}_p$, multiplication by t or t^2 given the values of $T' \cdot x_1$ or $L \cdot y_1$ is less expensive than the full multiplications $T' \cdot x_2$ or $L \cdot y_2$ for generic (x_2, y_2) , decreasing the pairing cost by $2\mathbf{m} + 10\mathbf{a}$.

Table 1. Binary Miller loop cost in 2 parallel pairings.

Technique	Cost	Ratio
Costello <i>et al.</i> [5]	$75\mathbf{m} + 5\mathbf{s} + 131\mathbf{a}$	1
Ours (plain)	$59\mathbf{m} + 139\mathbf{a}$	≈ 0.747
Ours (entangled)	$57\mathbf{m} + 129\mathbf{a}$	≈ 0.722

The performance improvements brought about by the techniques we propose are summarized on Table 1, assuming $1\mathbf{s} \approx 0.8\mathbf{m}$ and essentially ignoring \mathbf{a} . Our proposed variant of the parallel reduced Tate pairing is shown in full detail as Algorithm A.1 in the Appendix.

4.2 Ternary-Order Pairings

The computation of the reduced Tate pairing $e_r(P, Q)$ of order $r = 3^n$ proceeds as described in Algorithm 4.2. Again, the tripling formulas in Jacobian coordinates have an exception when $y = 0$, but this can be handled in a similar fashion to the binary case. The difference in running time reveals no private information for the same reason, namely only public data is involved in the pairing computations.

Algorithm 4.2. Tate3(P, Q): basic reduced Tate pairing of order $r = 3^n$:

INPUT: points P, Q .

OUTPUT: $e_r(P, Q)$.

- 1: $f \leftarrow 1, V \leftarrow P$
 - 2: **for** $i \leftarrow 0$ **to** $n - 1$ **do**
 - 3: $f \leftarrow f^3 \cdot g_{V,V}(Q) \cdot g_{V,[2]V}(Q) / (g_{[2]V}(Q) \cdot g_{[3]V}(Q)), V \leftarrow [3]V$
 - 4: **end for**
 - 5: **return** $e_r(P, Q) \leftarrow f^{(p^2-1)/r}$
-

The most efficient tripling algorithm known for Jacobian coordinates is due to Bernstein and Lange [4]. Let $V = [X : Y : Z : T]$ and $[3]V = [X' : Y' : Z' : T']$ in the extended coordinate system as before. Then:

$$\begin{array}{lll}
 X_2 \leftarrow X^2; & Y_2 \leftarrow Y^2; & Y_4 \leftarrow Y_2^2; \\
 T_2 \leftarrow T^2; & M \leftarrow 3X_2 + a \cdot T_2; & M_2 \leftarrow M^2; \\
 D \leftarrow (X + Y_2)^2 - X_2 - Y_4; & F \leftarrow 6D - M_2; & F_2 \leftarrow F^2; \\
 W \leftarrow 2Y_2; & W' \leftarrow 2W & S \leftarrow 16Y_4; \\
 U \leftarrow (M + F)^2 - M_2 - F_2 - S; & U' \leftarrow S - U; & \\
 X' \leftarrow 4(X \cdot F_2 - W' \cdot U); & Y' \leftarrow 8Y \cdot (U \cdot U' - F \cdot F_2); & \\
 Z' \leftarrow (Z + F)^2 - T - F_2; & T' \leftarrow (Z')^2; &
 \end{array}$$

The cost is $6\mathbf{M} + 10\mathbf{S} + 25\mathbf{A}$. This algorithm yields intermediate values that are useful in the calculation of a function equivalent to $g_{V,V}(Q) \cdot g_{V,[2]V}(Q) / (g_{[2]V}(Q) \cdot g_{[3]V}(Q))$, namely $\tilde{g}_3(V, Q) := (M \cdot h + d) \cdot (U' \cdot h + F' \cdot d) \cdot (W' \cdot h + F)^* \cdot R \cdot h_3^*$ when $[3]V \neq O$; $\tilde{g}_3(V, Q) := (M \cdot h + d) \cdot L^*$ when $[3]V = O$ but $V \neq O$; or simply $\tilde{g}_3(V, Q) := 1$ when $V = O$, where $h := T \cdot x - X$, $d := W - L \cdot y$, $h_3 := T' \cdot x - X'$, $F' := 2F$, $L \leftarrow ((Y + Z)^2 - Y_2 - T) \cdot T$, and $R := F \cdot T^*$.

One can further optimize the computation of a function $\hat{g}_3(V, Q)$ equivalent to $\tilde{g}_3(V, Q)$ in a similar fashion to what was done for the binary case. First, the expression $T' \cdot x - X'$ that occurs at a certain step will play the role of $T \cdot x - X$ at the next step, so one can simply store it from one step to the next and thus save $1\mathbf{M}$. Second, one can show that all R and L^* factors that appear in the definition of $\tilde{g}_3(V, Q)$ are irrelevant to the pairing value, and can be omitted. We defer the details for the extended version of this paper [16].

Consequently, the parabola function construction can be completed by computing only L and F' as above at a cost $1\mathbf{M} + 1\mathbf{S} + 4\mathbf{A}$. After initializing $h \leftarrow T \cdot x - X$ before Miller's loop at a cost of $1\mathbf{M}$ per pairing, the value g of function $\tilde{g}_3(V, Q)$ can be evaluated as

$$\begin{aligned} d &\leftarrow W - L \cdot y; & g &\leftarrow (M \cdot h + d) \cdot (U' \cdot h + F' \cdot d) \cdot (W' \cdot h + F); \\ h &\leftarrow T' \cdot x - X'; & g &\leftarrow g \cdot h^*; \end{aligned}$$

at a cost $9\mathbf{M} + 5\mathbf{A}$ per step of Miller's loop, except at the final step, when it is simply $g \leftarrow M \cdot h + d$.

Finally, updating f at each step as $f \leftarrow f^3 \cdot \tilde{g}_3(V, Q)$ incurs a cost $1\mathbf{C}$ to compute the complex cube f^3 , plus $1\mathbf{M}$ to compute $f^3 \cdot \tilde{g}_3(V, Q)$ from f^3 and $\tilde{g}_3(V, Q)$. Therefore, the proposed variant has the following overall cost per step, where again the shared part is amortized among parallel pairings that share the same first argument:

- (shared) cost of point tripling and parabola function construction: $6\mathbf{M} + 10\mathbf{S} + 25\mathbf{A} + 1\mathbf{M} + 1\mathbf{S} + 4\mathbf{A} = 43\mathbf{m} + 126\mathbf{a}$;
- (individual) cost of parabola function evaluation and accumulation: $9\mathbf{M} + 5\mathbf{A} + 1\mathbf{C} = 32\mathbf{m} + 2\mathbf{s} + 66\mathbf{a}$.

By comparison, the Costello *et al.* [5] technique has the following costs:

- (shared) cost of point tripling and construction of the parabola functions: $19\mathbf{M} + 6\mathbf{S} + 6\mathbf{s} + 15\mathbf{a} = 69\mathbf{m} + 6\mathbf{s} + 128\mathbf{a}$;
- (individual) cost of evaluating the parabola functions and accumulating the results: $10\mathbf{M} + 2\mathbf{S} + 4\mathbf{a} = 34\mathbf{m} + 60\mathbf{a}$.

The performance improvements brought about by the techniques we propose are summarized on Table 2. Our proposed variant of the parallel reduced Tate pairing is shown in full detail in the Appendix as Algorithm A.2.

Table 2. Ternary Miller loop cost in 2 parallel pairings.

Technique	Cost	Ratio
Costello <i>et al.</i> [5]	137m + 6s + 248a	1
Ours	107m + 4s + 258a	≈ 0.777

5 Discrete Logarithm Computation

Let ℓ^e be one of the prime-power factors of $p + 1 = 2^m \cdot 3^n$. Let $\mu_{\ell^e} \subset \mathbb{F}_{p^2}$ be the set of ℓ^e -th roots of unity in \mathbb{F}_{p^2} , i.e. $\mu_{\ell^e} := \{v \in \mathbb{F}_{p^2} \mid v^{\ell^e} = 1\}$. Inverting in μ_{ℓ^e} is a mere conjugation, $(a + bi)^{-1} = a - bi$ since the norm is 1. The Pohlig-Hellman method (Algorithm 5.1) to compute the discrete logarithm of $c \in \mu_{\ell^e}$ requires solving an equation of the form:

$$r_k^{\ell^{e-1-k}} = s^{d_k}$$

where $s = g^{\ell^{e-1}}$ has order ℓ and, for $k = 0, \dots, e - 1$, $d_k \in \{0, \dots, \ell - 1\}$ is an ℓ -ary digit, $r_0 = c$, and r_{k+1} depends on r_k and d_k .

Algorithm 5.1. Basic Pohlig-Hellman discrete logarithm algorithm

INPUT: generator $g \in \mu_{\ell^e}$, challenge $c \in \mu_{\ell^e}$.

OUTPUT: $d := \log_g c$, i.e. $g^d = c$.

- 1: $s \leftarrow g^{\ell^{e-1}}$ // NB: $s^\ell = 1$
 - 2: $d \leftarrow 0, r_0 \leftarrow c$
 - 3: **for** $k \leftarrow 0$ **to** $e - 1$ **do**
 - 4: $v_k \leftarrow r_k^{\ell^{e-1-k}}$
 - 5: find $d_k \in \{0, \dots, \ell - 1\}$ such that $v_k = s^{d_k}$
 - 6: $d \leftarrow d + d_k \ell^k, r_{k+1} \leftarrow r_k \cdot g^{-\ell^k d_k}$
 - 7: **end for** // NB: $g^d = c$
 - 8: **return** d
-

Assuming that $g^{-\ell^k}$ is precomputed and stored for all k as a by-product of the computation of s , the naive strategy to obtain the discrete logarithm requires repeatedly computing the exponential $r_k^{\ell^{e-1-k}}$ at the cost of $e - 1 - k$ raisings to the ℓ , then solving a small discrete logarithm instance in a subgroup of order ℓ to get one ℓ -ary digit, then clearing that digit in the exponent of r_k at a cost not exceeding ℓ multiplications to obtain r_{k+1} . The overall cost is thus $O(e^2)$.

It turns out that this strategy is far from optimal, as pointed out by Shoup [13, Chap. 11]. The crucial task is to obtain the sequence $r_0^{\ell^{e-1}}, r_1^{\ell^{e-2}}, r_2^{\ell^{e-3}}, \dots, r_{e-1}^{\ell^0}$ in this order, since each r_k depends on the previous one.

In our case, the set of vertices is $\{\Delta_{jk} \mid j + k \leq e - 1\}$ where $\Delta_{jk} := r_k^{\ell^j}$. Each vertex has either two downward outgoing edges, or no edges at all. Vertices Δ_{jk}

with $j + k > e - 1$ have two edges: a left edge $\Delta_{jk} \rightarrow \Delta_{j+1,k}$ that models raising the source vertex to the ℓ -th power to yield the destination vertex, $r_k^{\ell^{j+1}} \leftarrow (r_k^{\ell^j})^\ell$, and a right edge $\Delta_{jk} \rightarrow \Delta_{j,k+1}$ that models clearing the $(j + k)$ -th digit in the exponent of the source vertex, $r_{k+1}^{\ell^j} \leftarrow r_k^{\ell^j} \cdot g^{-\ell^{(j+k)}d_k}$. Vertices Δ_{jk} with $j + k = e - 1$ are leaves since they have no outgoing edges.

De Feo *et al.* [6, Eq. 5] describe an $O(e^2)$ dynamic programming algorithm that computes the cost of an optimal subtree of Δ with root at Δ_{00} and covering all leaves. If the cost of traversing a left or right edge is p or q respectively, and the cost of an optimal subtree of k edges is $C_{p,q}(k)$, their algorithm is based on the relations $C_{p,q}(1) = 0$ and $C_{p,q}(k) = \min_{1 \leq j < k} (C_{p,q}(j) + C_{p,q}(k - j) + (k - j)p + jq)$ for $k > 1$.

The naive dynamic programming approach is to store the values of $C_{p,q}(k)$ for $k = 1 \dots e$, invoking the above relation $k - 1$ times at each step to find the corresponding minimum, for a total $e(e - 1)/2$ invocations, hence the $O(e^2)$ cost. However, because $C_{p,q}(k)$ has no local minimum other than the single global minimum (or two adjacent, equivalent copies of the global minimum at worst), one can find that minimum with a variant of binary search that compares two consecutive values near the middle of the search interval $[1 \dots k - 1]$ and then halves that interval. This yields the $O(e \log e)$ Algorithm 5.2, which computes $C_{p,q}(k)$ and the structure of the optimal traversal strategy by storing the values of j above that attain the minimum at each step.

Algorithm 5.2. OptPath(p, q, e): optimal subtree traversal path

INPUT: p, q : left and right edge traversal cost; e : number of leaves of Δ .
 OUTPUT: P : optimal traversal path

```

1: Define  $C[1 \dots e]$  as an array of costs and  $P[1 \dots e]$  as an array of indices.
2:  $C[1] \leftarrow 0, P[1] \leftarrow 0$ 
3: for  $k \leftarrow 2$  to  $e$  do
4:    $j \leftarrow 1, z \leftarrow k - 1$ 
5:   while  $j < z$  do
6:      $m \leftarrow j + \lfloor (z - j)/2 \rfloor, u \leftarrow m + 1$ 
7:      $t_1 \leftarrow C[m] + C[k - m] + (k - m) \cdot p + m \cdot q$ 
8:      $t_2 \leftarrow C[u] + C[k - u] + (k - u) \cdot p + u \cdot q$ 
9:     if  $t_1 \leq t_2$  then
10:       $z \leftarrow m$ 
11:     else
12:       $j \leftarrow u$ 
13:     end if
14:   end while
15:    $C[k] \leftarrow C[j] + C[k - j] + (k - j) \cdot p + j \cdot q, P[k] \leftarrow j$ 
16: end for
17: return  $P$ 

```

5.1 Discrete Logarithm Computation Cost

The cost of an optimal strategy depends on the individual costs of traversing a left edge and a right edge. We now show that, because of our proposed reverse basis decomposition technique, the total cost of discrete logarithm computation is drastically reduced. A left edge traversal represents the computation $r_k^{\ell^{j+1}} \leftarrow (r_k^{\ell^j})^\ell$ at a cost $w\mathbf{S} \approx 2w\mathbf{m}$ in the binary case and $w\mathbf{C} = w(2\mathbf{m} + \mathbf{1s}) \approx 2.8w\mathbf{m}$ in the ternary case, with windows of size w .

A right edge traversal represents the computation $r_{k+1}^{\ell^j} \leftarrow r_k^{\ell^j} \cdot g^{-\ell^{(j+k)}d_k}$, which can be performed via table lookup $r_{k+1}^{\ell^j} \leftarrow r_k^{\ell^j} \cdot T[j+k][d_k]$ where $T[u][d] := g^{-\ell^u \cdot d}$. Since $j+k \leq e-1$, the table size is $(e/w) \cdot \ell^w$ field elements. However, no more than a single multiplication is incurred regardless of ℓ , e , or w , namely, $1\mathbf{M} \approx 3\mathbf{m}$. When w is very small, avoiding the multiplication for $d_k = 1$ noticeably reduces the running time and requires fewer table entries. Moreover, the table is *fixed* with the reverse basis decomposition technique, because $g = e(P_B, Q_B)^{\deg \phi_A}$, or $g = e(P_A, Q_A)^{\deg \phi_B}$, thus incurring no table building cost at running time for each newly generated key. Even the simple discrete logarithm instances at the leaves only incur $O(\ell)$ lookups on the same table, since $s^{d_k} = T[e-1][d_k]^*$.

Algorithm 5.3 summarizes the proposed technique, combining Shoup’s RDL algorithm [13, Sect. 11.2.3] with the optimal divide-and-conquer strategy of De Feo *et al.* and the efficient table lookup enabled by reverse basis decomposition.

Algorithm 5.3. $\text{Traverse}(r, j, k, z, P, T, d)$

INPUT: r : value of root vertex Δ_{jk} , i.e. $r := r_k^{\ell^j}$; j, k : coordinates of root vertex Δ_{jk} ;
 z : number of leaves in subtree rooted at Δ_{jk} ; P : traversal path; T : lookup table.

OUTPUT: d : digits (base ℓ) of $\log_g r_0$.

REMARK: initial call is $\text{Traverse}(r_0, 0, 0, e, P, T, d)$.

```

1: if  $z > 1$  then
2:    $t \leftarrow P[z]$  //  $z$  leaves:  $t$  to the left exp,  $z - t$  to the right
3:    $r' \leftarrow r^{\ell^{z-t}}$  // go left  $(z - t)$  times
4:    $\text{Traverse}(r', j + (z - t), k, t, P, T)$ 
5:    $r' \leftarrow r \cdot \prod_{h=k}^{k+t-1} T[j+h][d_h]$  // go right  $t$  times
6:    $\text{Traverse}(r', j, k + t, z - t, P, T)$ 
7: else // leaf
8:   find  $t \in \{0, \dots, \ell - 1\}$  such that  $r = T[e - 1][t]^*$ 
9:    $d_k \leftarrow t$  // recover  $k$ -th digit  $d_k$  of the discrete logarithm from  $r = s^{d_k}$ 
10: end if

```

The resulting improvements are substantial. For discrete logs in μ_{2372} , the optimal cost is $\approx 4958.4\mathbf{m}$ with windows of size $w = 1$, $\approx 3127.9\mathbf{m}$ with windows of size $w = 3$, and $\approx 2103.7\mathbf{m}$ with windows of size $w = 6$. For discrete logs in μ_{3239} , the optimal cost is $\approx 4507.6\mathbf{m}$ with windows of size $w = 1$, $\approx 2638.1\mathbf{m}$ with windows of size $w = 3$, and $\approx 1739.8\mathbf{m}$ with windows of size $w = 6$.

Tradeoffs are also possible. Instead of being a matrix of size $(e/w) \cdot \ell^w$, the lookup table could be restricted to a single array $T_1[u] := g^{-\ell^u}$ of (e/w) entries, by computing $T_1[u]^d = g^{-\ell^u \cdot d}$ on demand using an optimal multiplication chain for cyclotomic exponentiation. For instance, discrete logs in $\mu_{2^{372}}$ with windows size $w = 3$ would require a table of size 124 at an average cost $\approx 4453.9\mathbf{m}$. For comparison, the best results reported in [5, Sect. 5] are $5320\mathbf{m} + 3349\mathbf{s} \approx 8271.6\mathbf{m}$ for discrete logs in $\mu_{2^{372}}$ and $5320\mathbf{m} + 3349\mathbf{s} \approx 7999.2\mathbf{m}$ for discrete logs in $\mu_{3^{239}}$, both with windows of size $w = 3$, which is optimal in that technique; increasing the window size actually causes a cost increase.

Table 3 summarizes the gains our technique makes possible and compares them against the results from Costello *et al.*, in terms of both the raw number of multiplications in the base field and the ratio between our results and theirs. We recall that no side-channel security concern arises from this technique, since all information involved in the processing is public.

Table 3. Discrete logarithm computation costs (assuming $\mathbf{s} \approx 0.8\mathbf{m}$)

Group	Costello <i>et al.</i> [5]	Ours, $w = 1$ (ratio)	Ours, $w = 3$ (ratio)	Ours, $w = 6$ (ratio)
$\mu_{2^{372}}$	8271.6 \mathbf{m}	4958.4 \mathbf{m} (0.60)	3127.9 \mathbf{m} (0.39)	2103.7 \mathbf{m} (0.25)
$\mu_{3^{239}}$	7999.2 \mathbf{m}	4507.6 \mathbf{m} (0.56)	2638.1 \mathbf{m} (0.33)	1739.8 \mathbf{m} (0.22)

6 Point Tripling on Montgomery Curves

Multiplication by 3^n , be it as a cofactor in the case of the 2^m torsion or as a tool to test linear independence in the 3^n torsion, is a computationally expensive operation. We describe in Algorithm 6.1 an improved method for point tripling on Montgomery curves that, though modest, directly addresses this bottleneck.

Algorithm 6.1. Improved tripling on the Montgomery curve $By^2 = x^3 + Ax^2 + x$

INPUT: $P = (x, z)$: Montgomery curve point in xz representation.

OUTPUT: $[3]P = (x', z')$.

- 1: $t_1 \leftarrow x^2$; $t_2 \leftarrow z^2$;
 - 2: $t_3 \leftarrow t_1 + t_2$
 - 3: $t_4 \leftarrow 2A \cdot ((x + z)^2 - t_3) + t_3$
 - 4: $t_3 \leftarrow (t_1 - t_2)^2$;
 - 5: $t_1 \leftarrow (t_1 \cdot t_4 - t_3)^2$; $t_2 \leftarrow (t_2 \cdot t_4 - t_3)^2$;
 - 6: $x' \leftarrow x \cdot t_2$; $z' \leftarrow z \cdot t_1$;
 - 7: **return** (x', z')
-

The cost of our tripling is $5\mathbf{M}+6\mathbf{S}+7\mathbf{A}$ (or one less multiplication in scenarios where the curve coefficient A can be carefully chosen and fixed) with 4 ancillary variables, not counting the left shift (multiplication by 2) which costs no more than an addition but can be precomputed for a given curve. It is less expensive than the previously best tripling algorithm in the literature, which only attains $6\mathbf{M}+5\mathbf{S}+7\mathbf{A}$ with 8 ancillary variables [11, Appendix B]. Note that this tripling algorithm can be employed in the key (de)compression operations since they do not require the curve coefficient A to be in projective form. The projective version is only required in the computation of 3^n -isogenies, where field inversions can be avoided if the projective form is adopted. That is the case of the tripling formula by Faz-Hernández *et al.* [7], which costs $7\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$.

7 Implementation and Experimental Results

Our improved key compression and decompression techniques have been implemented on top of the SIDH C library [10] to make full-fledge key exchange available. We left the previous (de)compression functions in the new version to enable replicating the experiments and comparisons.

Since we only process public information (compression and decompression of public keys), side-channel attacks are not an issue, and faster non-isochronous algorithms like the extended Euclidean algorithm have been adopted.

Table 4. Benchmarks in cycles on an Intel Core i5 clocked at 2.9 GHz (clang compiler with `-O3` flag, and $s = m$ in this implementation).

Operations	2^m -torsion ($w = 2$)			3^n -torsion ($w = 1$)		
	SIDH v2.0 [5]	Ours	Ratio	SIDH v2.0 [5]	Ours	Ratio
Basis generation	24497344	1690452	14.49	20632876	17930437	1.15
Discrete log.	6206319	2776568	2.24	4710245	3069234	1.53
Pairing phase	33853114	25755714	1.31	39970384	30763841	1.30
Compression	78952537	38755681	2.04	78919488	61768917	1.28
Decompression	30057506	9990949	3.01	25809348	23667913	1.09

The initial public curve is the usual supersingular curve $E_0 : y^2 = x^3 + x$ defined over \mathbb{F}_{p^2} where $p = 2^{372}3^{239} - 1$. It is worth mentioning that before applying our (de)compression techniques, the SIDH v2.0 library was first modified to perform Alice’s key generation with both points P_A and Q_A defined over the extension $E_0(\mathbb{F}_{p^2}) \setminus E_0(\mathbb{F}_p)$ instead of defining P_A in the base field as suggested in [5]. The approach in [5] starts with point $P_A = (x, y) \in E_0(\mathbb{F}_p)$ over the base field and then applies the distortion map τ to get a linearly independent point $Q_A = \tau(P_A) = (-x, iy)$ lying on the trace zero group.

This optimization cannot be combined with our techniques because using distortion maps on binary torsions only gives a basis $\langle P_A, \tau(P_A) \rangle = E_0[2^{m-1}]$ of a smaller group of order $2^{2(m-1)}$, and in this case the images of P_A and $Q_A = \tau(P_A)$ under Bob's isogeny consequently generate a smaller torsion as well, i.e. $\langle \phi_B(P_A), \phi_B(Q_A) \rangle = E_B[2^{m-1}]$. In particular, the reverse basis decomposition technique combined with entangled basis would not work since an entangled basis generates the full 2^m -torsion, and this basis cannot be converted to a basis of a smaller torsion, i.e. the change of basis matrix in Eq. 3 would not exist. Therefore, we selected the new points

$$P_A := 3^{239} \cdot (5 + i, \sqrt{(5 + i)^3 + 5 + i}) \in E_0(\mathbb{F}_{p^2}) \setminus E_0(\mathbb{F}_p)$$

and $Q_A := \tau(P_A) \in E_0(\mathbb{F}_{p^2}) \setminus E_0(\mathbb{F}_p)$. Points P_B and Q_B are the ones in [5] since for ℓ^n torsions with ℓ odd, distortion maps do generate the full group $E_0[\ell^n]$ and P_B can be kept over the base field. For the discrete logarithms we set $w = 2$ for the binary case and $w = 1$ for the ternary one. Table 4 summarizes our experimental results with respect to the previous state-of-the-art implementation.

8 Conclusion

In this paper we proposed a range of new algorithms and techniques to speed up the supersingular isogeny Diffie-Hellman. For example, in the 2^m -torsion using $w = 2$ for the discrete logarithms, the key compression is about $2\times$ faster than the SIDH library and decompression achieves a factor of $3\times$, while the basis generation itself is nearly $14.5\times$ faster. The main bottleneck now, by far, is the pairing phase, that takes about 25.8M cycles against 1.7M for basis generation and 2.8M for the discrete logarithm phase. It is worthwhile to point out that the techniques of entangled basis generation and the optimal strategy applied to solve smooth-order discrete logarithms not only set up new speed records for those tasks, but might find new applications in different contexts in cryptography. We leave the possibility of extending the new entangled basis generation technique to non-binary torsions as an open problem.

Acknowledgment. J. Doliskani and G. Pereira were supported by NSERC, CryptoWorks21, and Public Works and Government Services Canada. M. Simplicio was supported by Brazilian National Council for Scientific and Technological Development (CNPq) under grant 301198/2017-9. M. Simplicio, P. Barreto and G. Zanon were partially supported by the joint São Paulo Research Foundation (FAPESP) / Intel Research grant 2015/50520-6 “Efficient Post-Quantum Cryptography for Building Advanced Security Applications.” M. Simplicio and P. Barreto are also partially supported by the São Paulo Research Foundation (FAPESP) under grant 13/25977-7.

A Pairing Algorithms

<p>Algorithm A.1. Tate2($P, [Q_j], m$): reduced Tate pairing of order $r = 2^m$</p> <hr/> <p>INPUT: Curve $E: y^2 = x^3 + ax + b$ – Point $P = [X_P : Y_P : Z_P]$ on E of order 2^m – t points $Q_j = [X_{Q_j} : Y_{Q_j} : Z_{Q_j}]$ on E, $Z_{Q_j} \in \{0, 1\}$ OUTPUT: List of t values $e_{2^m}(P, Q_j)$</p> <hr/> <pre> 1: $X \leftarrow X_P; Y \leftarrow Y_P; Z \leftarrow Z_P; T \leftarrow Z^2$ ▷ NB: the following operations are in \mathbb{F}_{p^2} 2: for $j \leftarrow 0$ to $t - 1$ do 3: $f_j \leftarrow 1; h_j \leftarrow T \cdot X_{Q_j} - X;$ 4: end for 5: for $k \leftarrow 0$ to $m - 1$ do ▷ <i>point doubling and line function construction:</i> 6: $X_2 \leftarrow X^2; Y_2 \leftarrow Y^2; Y_4 \leftarrow Y_2^2$ 7: $M \leftarrow 3X_2 + a \cdot T^2$ 8: $S \leftarrow 2((X + Y_2)^2 - X_2 - Y_4)$ 9: $X' \leftarrow M^2 - 2S$ 10: $Y' \leftarrow M \cdot (S - X') - 8Y_4$ 11: $Z' \leftarrow (Y + Z)^2 - Y_2 - T;$ 12: $T' \leftarrow (Z')^2; L \leftarrow Z' \cdot T; W \leftarrow 2Y_2$ 13: if $Z' = 0$ then // <i>exception for points in [2]E</i> 14: $X' \leftarrow 0; Y' \leftarrow 1$ 15: end if ▷ <i>line function evaluation and accumulation:</i> 16: for $j \leftarrow 0$ to $t - 1$ do 17: if $Z' \neq 0$ then 18: $g \leftarrow M \cdot h_j + W - L \cdot Y_{Q_j}$ 19: $h_j \leftarrow T' \cdot X_{Q_j} - X'$ 20: $g \leftarrow g \cdot h_j^*$ 21: else 22: $g \leftarrow h_j;$ 23: end if 24: $f_j \leftarrow f_j^2; f_j \leftarrow f_j \cdot g;$ 25: end for 26: $X \leftarrow X'; Y \leftarrow Y';$ 27: $Z \leftarrow Z'; T \leftarrow T';$ 28: end for 29: return $[(Z_{Q_j} \neq 0) f_j^{(p^2-1)/r} : 1 \mid j = 0 \dots t - 1]$ </pre>	<p>Algorithm A.2. Tate3($P, [Q_j], n$): reduced Tate pairing of order $r = 3^n$</p> <hr/> <p>INPUT: Curve $E: y^2 = x^3 + ax + b$ – Point $P = [X_P : Y_P : Z_P]$ on E of order 3^n – t points $Q_j = [X_{Q_j} : Y_{Q_j} : Z_{Q_j}]$ on E, $Z_{Q_j} \in \{0, 1\}$ OUTPUT: List of t values $e_{3^n}(P, Q_j)$</p> <hr/> <pre> 1: $X \leftarrow X_P; Y \leftarrow Y_P; Z \leftarrow Z_P; T \leftarrow Z^2;$ ▷ NB: the following operations are in \mathbb{F}_{p^3} 2: for $j \leftarrow 0$ to $t - 1$ do 3: $f_j \leftarrow 1; h_j \leftarrow T \cdot X_{Q_j} - X;$ 4: end for 5: for $k \leftarrow 0$ to $n - 1$ do ▷ <i>point tripling and parabola function construction:</i> 6: $X_2 \leftarrow X^2; Y_2 \leftarrow Y^2; Y_4 \leftarrow Y_2^2; T_2 \leftarrow T^2;$ 7: $M \leftarrow 3X_2 + a \cdot T_2; M_2 \leftarrow M^2$ 8: $D \leftarrow (X + Y_2)^2 - X_2 - Y_4;$ 9: $F \leftarrow 6D - M_2; F_2 \leftarrow F^2$ 10: $W \leftarrow 2Y_2; W' \leftarrow 2W; S \leftarrow 16Y_4$ 11: $U \leftarrow (M + F)^2 - M_2 - F_2 - S; U' \leftarrow S - U$ 12: $X' \leftarrow 4(X \cdot F_2 - W' \cdot U)$ 13: $Y' \leftarrow 8Y \cdot (U' \cdot U' - F \cdot F_2)$ 14: $Z' \leftarrow (Z + F)^2 - T - F_2; T' \leftarrow (Z')^2$ 15: $L \leftarrow ((Y + Z)^2 - Y_2 - T) \cdot T; F' \leftarrow 2F;$ 16: if $Z' = 0$ then // <i>exception for points in [3]E</i> 17: $X' \leftarrow 0; Y' \leftarrow 1$ 18: end if ▷ <i>parabola function evaluation and accumulation:</i> 19: for $j \leftarrow 0$ to $t - 1$ do 20: $d \leftarrow W - L \cdot Y_{Q_j};$ 21: if $Z' \neq 0$ then 22: $g \leftarrow (M \cdot h + d)(U' \cdot h + F' \cdot d)(W' \cdot h + F) \cdot$ 23: $h \leftarrow T' \cdot X_{Q_j} - X'; g \leftarrow g \cdot h^*$ 24: else 25: $g \leftarrow M \cdot h + d$ 26: end if 27: $f \leftarrow f^3; f \leftarrow f \cdot g$ 28: end for 29: $X \leftarrow X'; Y \leftarrow Y'; Z \leftarrow Z'; T \leftarrow T'$ 30: end for 31: return $[(Z_{Q_j} \neq 0) f_j^{(p^3-1)/r} : 1 \mid j = 0 \dots t - 1]$ </pre>
--	---

References

1. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key compression for isogeny-based cryptosystems. In: Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, pp. 1–10. ACM (2016)
2. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–369. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_23
3. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 967–980. ACM (2013)
4. Bernstein, D.J., Lange, T.: Analysis and optimization of elliptic-curve single-scalar multiplication. In: Finite Fields and Applications: Proceedings of Fq8, Number 461 in Contemporary Mathematics, pp. 1–18. American Mathematical Society, Providence (2008)

5. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient compression of SIDH public keys. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 679–706. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_24
6. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.* **8**(3), 209–247 (2014)
7. Faz-Hernández, A., López, J., Ochoa-Jiménez, E., Rodríguez-Henríquez, F.: A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. *Cryptology ePrint Archive*, Report 2017/1015 (2017)
8. Husemöller, D.: *Elliptic Curves: Graduate Texts in Mathematics*, vol. 111, 2nd edn. Springer, New York (2004). <https://doi.org/10.1007/978-0-387-09494-6>
9. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-319-56620-7_24
10. MS SIDH team: SIDH v2.0 (2017). <https://github.com/Microsoft/PQCrypto-SIDH>
11. Subramanya Rao, S.R.: Three dimensional montgomery ladder, differential point tripling on montgomery curves and point quintupling on weierstrass' and edwards curves. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2016. LNCS, vol. 9646, pp. 84–106. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31517-1_5
12. Schaefer, E., Stoll, M.: How to do a p -descent on an elliptic curve. *Trans. Am. Math. Soc.* **356**(3), 1209–1231 (2004)
13. Shoup, V.: *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, Cambridge (2005)
14. Silverman, J.H.: *The Arithmetic of Elliptic Curves: Graduate Texts in Mathematics*, vol. 106, 2nd edn. Springer, New York (2009). <https://doi.org/10.1007/b97292>
15. Spiegel, M.R., Liu, J.: *Mathematical Handbook of Formulas and Tables*. Schaum's Outline Series, 2nd edn. McGraw-Hill, New York (1999)
16. Zanon, G.H.M., Simplicio Jr., M.A., Pereira, G.C.C.F., Doliskani, J., Barreto, P.S.L.M.: Faster isogeny-based compressed key agreement. Technical report, *Cryptology ePrint Archive*, Report 2017/1143 (2017)

Lattice-Based Cryptography



Practical Implementation of Ring-SIS/LWE Based Signature and IBE

Pauline Bert^(✉), Pierre-Alain Fouque, Adeline Roux-Langlois,
and Mohamed Sabt

Univ Rennes, CNRS, IRISA, Rennes, France
`pauline.bert@irisa.fr`

Abstract. Lattice-based signature and Identity-Based Encryption are well-known cryptographic schemes, and having both efficient and provable secure schemes in the standard model is still a challenging task in light of the current NIST post-quantum competition. We address this problem in this paper by mixing standard IBE scheme, à la ABB (EUROCRYPT 2010) on Ring-SIS/LWE assumptions with the efficient trapdoor of Peikert and Micciancio (EUROCRYPT 2012) and we provide an efficient implementation. Our IBE scheme is more efficient than the IBE scheme of Ducas, Lyubashevsky and Prest based on NTRU assumption and is based on more standard assumptions. We also describe and implement the underlying signature scheme, which is provably secure in the standard model and efficient.

Keywords: Lattice · Signature · IBE · Software implementation
Ring-LWE/SIS

1 Introduction

The concept of Identity Based Encryption (IBE) was defined by Shamir [Sha85]. It is considered as an alternative to the classical Public Key Encryption (PKE), often requiring a dedicated infrastructure. Indeed, the public key related to a person is simply its identity, such as her email address or her social security number, and the associated private key is generated by a trusted authority using a master public key. Thus, IBE hugely simplifies keys generation and distribution in a multi-user system. The first IBE constructions appeared in [BF01, Coc01], and were based respectively on bilinear maps and on quadratic residue assumptions.

Since the work of Shor in 1994 [Sho97], the hardness of such number theoretic assumptions is extremely reduced when faced to a quantum computer. This has motivated many research work attempting to achieve quantum security. The first supposedly post-quantum IBE scheme, which was introduced in 2008 by Gentry et al. [GPV08], was based on hard lattice problems. Since then, many improvements have been proposed [CHKP10, ABB10, DLP14].

Lattice-based cryptography. Lattice-based cryptography starts with the work of Ajtai [Ajt96], and uses hard problems on lattices as the foundation of secure cryptographic constructions. A lattice is an infinite arrangement of regularly spaced points, and it can be generated as the set of all linear combinations of m independent vectors in \mathbb{R}^n , called a basis. One fundamental hard problem on lattices is the Shortest Vector Problem (SVP): given some basis, find the shortest non zero vector of the lattice. Lattice-based cryptography is based on the assumption that this problem and its variants are hard problems even for an approximation factor polynomial in the dimension of the lattice.

Lattice-based cryptographic constructions are mainly based on two well known problems: the Small Integer Solution problem (SIS) and its Inhomogeneous variant (ISIS) [Ajt96], and the Learning With Errors problem (LWE) introduced by Regev [Reg05]. In particular, the ISIS problem consists in finding a short vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{Ax} = \mathbf{u} \pmod{q}$, given an uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and some $\mathbf{u} \in \mathbb{Z}_q^n$. Ajtai and Regev gave reductions from worst-case lattice problems to the average case LWE and SIS problems.

Few years later, structured variants of the LWE and SIS problems were proposed [Mic07, SSTX09], called Ring-SIS and Ring-LWE. These problems are preferred in practice, since they enjoy smaller storage and faster operations. There also exist reductions from worst-case ideal lattice problems to these structured variants [LPR10, SSTX09, LM06, PR06]. These two problems can be used to construct many basic cryptographic primitives such as PKE (adapting the schemes from [Reg05, GPV08]) and signatures [Lyu12, DDLL13, DM14].

Lattice Trapdoors. To construct IBE or Attribute Based Encryption (ABE), we can use trapdoors for the SIS problem. Initially described by Ajtai [Ajt96], a trapdoor $\mathbf{T}_\mathbf{A} \in \mathbb{Z}^{m \times m}$ for $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is a short basis of vectors satisfying $\mathbf{T}_\mathbf{A} \cdot \mathbf{A} = \mathbf{0} \pmod{q}$, generated together with \mathbf{A} . Given only \mathbf{A} , it is hard to find such a short basis but, with the knowledge of $\mathbf{T}_\mathbf{A}$ it is easy to invert the SIS (and the ISIS) problem. Trapdoors constructions were improved by [AP09], and then described for ideal lattices in [SSTX09].

Micciancio and Peikert [MP12] proposed a new construction allowing a faster inversion of the ISIS problem, by reducing it to the inversion of a smaller problem for some structured gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times l}$. The matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is generated with its associated short basis $\mathbf{R} \in \mathbb{Z}^{(m-l) \times l}$, as $\mathbf{A} = (\mathbf{A}' \mid \mathbf{HG} - \mathbf{A}'\mathbf{R})$ where $\mathbf{A}' \in \mathbb{Z}_q^{n \times (m-l)}$ is uniformly sampled, and $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ is an invertible element. In [GM18], Micciancio and Genise improved the inversion of the ISIS problem in the ring variant with an arbitrary modulus.

Lattice-based IBE. The first lattice-based IBE [GPV08] relies on the Dual-Regev encryption scheme. In Dual-Regev, a public key consists in a vector $\mathbf{u} = \mathbf{Ax} \pmod{q}$, for a short vector $\mathbf{x} \in \mathbb{Z}^m$ (which is the secret key). To build the IBE, an identity can be mapped to a public key via a hash function $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbb{Z}_q^n$ modeled as a random oracle. The master secret key of the IBE is a short trapdoor $\mathbf{T}_\mathbf{A}$, that makes it possible to extract a secret key \mathbf{x} associated to any id by inverting $\mathbf{Ax} = \mathcal{H}(id) \pmod{q}$.

This construction was later improved by removing the Random Oracle Model (ROM) [CHKP10, ABB10]. In ABB, a publicly computable matrix $\mathbf{A}_{id} = (\mathbf{A} \mid F(id)) \in \mathbb{Z}_q^{n \times (m+m')}$, is associated to an identity id , where $F(\cdot)$ is mapping identities to matrices in $\mathbb{Z}_q^{n \times m'}$. As a result, the secret key for an identity id is a short vector $\mathbf{x} \in \mathbb{Z}^{m+m'}$, such that $\mathbf{A}_{id}\mathbf{x} = \mathbf{u} \pmod q$. To find such an $\mathbf{x} = (\mathbf{x}_1^T \mid \mathbf{x}_2^T)^T$, it is only required to sample a short $\mathbf{x}_2 \in \mathbb{Z}^{m'}$, and to use $\mathbf{T}_{\mathbf{A}}$ to invert the following ISIS problem: find a small $\mathbf{x}_1 \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{x}_1 = \mathbf{u} - F(id)\mathbf{x}_2$. In 2014, Ducas et al. [DLP14] gave an NTRU variant of the GPV IBE scheme. In their work, the public key is built using NTRU lattices, which improves the efficiency of the scheme.

Our Contributions. In this paper, we provide the first software implementation of a standard model IBE based on the hardness of Ring-SIS/LWE. Our main goal is to show that IBE schemes can, and without sacrificing efficiency, guarantee better security by being on the standard model and relying on a classical assumption on lattice problems. We instantiate our implemented IBE scheme from the selective secure IBE scheme described in [ABB10] as well as from the recent variant of trapdoor described in [MP12, GM18]. We also describe and implement the underlying signature scheme that achieve a selective notion of unforgeability based on the hardness of Ring-SIS/LWE. We choose to implement these selective secure schemes in the standard model due to their efficiency and also their simplicity compared to other adaptive secure variants [ABB10, DM14]. Our constructions work over polynomial rings $R_q = \mathbb{Z}_q[x]/(x^n+1)$ with n a power of two and q a prime modulus congruent to 1 mod $2n$. This setting is wide-spread in ideal lattice cryptography due to its efficiency thanks to the Number Theoretic Transform (NTT) representation. Note that our complete implementation can be found at <https://github.com/lbibe/code>.

We provide our implementation as a general-purpose thread-safe C++ library. We take much care to write plain C++ and not to explicitly include highly specialized instructions, such as AVX2 and NEON, in order to ensure portability over several hardware architectures. Instead, we rely on the GCC 7.2 compiler to make vectorization, and a set of optimization methods and multi-threading have been introduced using a number of C++11 specific features. More importantly, we design our code to be modular and provide three software layers, each of which is of independent interest. The software layers are (1) the different Gaussian sampling techniques based on [DP16, GM18], (2) the Gaussian sampling for trapdoor lattice with arbitrary modulus using [MP12], and (3) the underlying cryptographic constructions (both IBE and the related signature).

We experimentally evaluate the performance of these two constructions. To our surprise, the obtained runtime is fast and even quite close to that of other schemes built in the ROM or/and using different security assumptions. We remind that our goal is not to provide the most efficient IBE or signature scheme, but to show that “good” security properties can still be practical for many scenarios. In Tables 1 and 2, we provide a comparison with state-of-the-art implementations of IBE and signature schemes. We note the security parameter as λ (classical bit security in Tables 1, 2, 3, 4, 5 and 6) and the lattice dimension as n .

Table 1. Timings for the different operations of IBE schemes: Setup (master key generation), Extract (user private key generation), encryption and decryption. Since Setup is performed only once, we provide the timing of only one single operation. Extract can measure how many users the system can manage. As for Encrypt/Decrypt, we give their throughput in KB per second.

Scheme	(λ, n)	Setup (ms)	Extract (ms)	Encrypt (KB/s)	Decrypt (KB/s)
BF-128 [Fou13]	(128, $-$)	$-$	0.55	4.10	6.19
DLP-14 [MSO17]	(80, 512)	4034	3.8	587	1405
This paper	(80, 1024)	1.67	4.02	230	1042

Table 2. Timings for the different operations of signature schemes: KeyGen (key generation), signature and verification. Since KeyGen is performed only once per user, we provide the timing of only one single operation. As for Sign/Verify, we give the timing as the number of sign/verify operations per second.

Scheme	(λ, n)	KeyGen (ms)	Sign (op/s)	Verify (op/s)
Falcon [FHK+18]	(195, 768) ^a	53.48	202	2685
This paper	(170, 1024)	0.96	540	21276

^acorresponds to NIST Security Level 2, since Level 1 only achieves 114 bits of security.

We obtain our results using dual-core Intel i7 2.6 GHz CPU with standard CPU benchmarks. Concerning the IBE schemes, we notice that our Setup (master key generation), compared to DLP-14, is (much) faster (we note that the DLP-14 Setup could now be improved using the recent results in Falcon [FHK+18]), while decryption and Extract are of the same order of magnitude. However, our encryption is twice slower. For the sake of completeness, we include the timings of the paring-based IBE scheme of Boneh-Franklin. As for the signature scheme, we compare our implementation with Falcon [FHK+18] that is the underlying signature scheme of [DLP14]. In our timings, we did not consider the phase of precomputations that we detail later in Sect. 5 (see Tables 5 and 6).

Related work. In [EBB13], the authors gave a software implementation of the trapdoor of Micciancio and Peikert [MP12] in both matrix and ring variants. They also included this trapdoor into the signature in the ROM of [GPV08]. Recently, in a concurrent work, a software implementation of the improvement lattice trapdoor [MP12, GM18] is given in [GPR+17, DDP+17], with application to respectively the GPV signature and the ABE scheme of [BGG+14].

The only lattice based IBE given with an implementation we are aware of, is the one of Ducas et al. [DLP14]. In [DLP14], the authors gave a proof-of-concept implementation, recently improved by [MSO17]. This IBE scheme is the IBE of [GPV08], working with structured NTRU lattices. The Gram-Schmidt norm of a NTRU basis is quite small and efficiently computed, then the Gaussian sampling using this basis outputs better quality vectors.

Conclusion and Open Problems. Our main contribution is a software implementation of a lattice-based IBE and signature scheme. Both constructions are proven secure in the standard model under the hardness of Ring-SIS/LWE. Our IBE is a ring-version of the selective ABE scheme, adapted using the MP12 trapdoor, we provide its underlying signature scheme and both security proofs.

We stress that our implementation has an efficiency comparable to NTRU based schemes in the ROM, even if we thought at first that a scheme on the standard model and based on Ring-SIS/LWE would be much less efficient. Then, we find it interesting to observe that even with the constraint on the choice of parameters and using a Gaussian sampling, those constructions, which are not using NTRU lattices, can also be efficient.

There are several open questions which would be interesting to answer. First, we choose to study the selective secure IBE scheme of ABB, and a signature scheme which achieve a selective notion of unforgeability. Both schemes can be improved to a better security by using the adaptive IBE scheme of ABB, and by looking at the standard secure signature of [DM14]. We also discuss in Sect. 2.4 the choice of the encoding hash function used in both schemes, which could be improved using the recent results of [LS18]. Finally, we want to modify NFLlib, which for now only allows us to use parameter q of size 30 or 62 bits. As discussed in Sect. 5, using a parameter q in between would optimize our choice of parameters. Note that using a Module variant, as in [DLL+17], could also be a solution.

2 Preliminaries

Notations. Let D be a distribution over some finite set S , then $x \leftarrow D$ means that x is chosen from the distribution D and $x \leftarrow U(S)$ denotes the sampling of a uniformly random element x from S . We denote column vectors and matrices in bold, respectively by bold lowercase (e.g. \mathbf{x}) and bold uppercase (e.g. \mathbf{A}). The euclidean norm of the vector \mathbf{x} is denoted by $\|\mathbf{x}\|$. The norm of a matrix $\|\mathbf{T}\| = \max_i \|\mathbf{t}_i\|$ is the maximum norm of its column vectors.

Lattices. An m -dimensional full-rank lattice Λ is a discrete additive subgroup of \mathbb{R}^m . A lattice is the set of all integer combinations of some linearly independent basis vectors, $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_m\} \in \mathbb{R}^{m \times m}$, $\Lambda(\mathbf{B}) = \{\sum_{i=1}^m z_i \mathbf{b}_i : z_i \in \mathbb{Z}\}$. For n a power of two, the polynomial ring $R = \mathbb{Z}[x]/(x^n + 1)$ is isomorphic to the integer lattice \mathbb{Z}^n , a polynomial $f = \sum_{i=0}^{n-1} f_i x^i$ in R corresponds to the integer vector of its coefficients (f_0, \dots, f_{n-1}) in \mathbb{Z}^n . The norm of a polynomial $\|f\|$ is the norm of its coefficients vector. For the rest of the paper we will work with polynomials over R , or $R_q = R/qR = \mathbb{Z}_q[x]/(x^n + 1)$ where q is a prime such that $q \equiv 1 \pmod{2n}$.

Gaussian distribution. The Gaussian function of center $\mathbf{c} \in \mathbb{R}^n$ and width parameter σ is defined as $\rho_{\sigma, \mathbf{c}}(\mathbf{x}) = \exp(-\pi \frac{\|\mathbf{x} - \mathbf{c}\|^2}{\sigma^2})$, for all $\mathbf{x} \in \mathbb{R}^n$. We can extend this definition to a positive definite covariance matrix $\Sigma = \mathbf{B}\mathbf{B}^T$:

$\rho_{\sqrt{\Sigma}, \mathbf{c}}(\mathbf{x}) = \exp(-\pi(\mathbf{x} - \mathbf{c})^T \Sigma^{-1}(\mathbf{x} - \mathbf{c}))$. The discrete Gaussian distribution over a lattice Λ is defined as $D_{\Lambda, \sigma, \mathbf{c}}(\mathbf{x}) = \frac{\rho_{\sigma, \mathbf{c}}(\mathbf{x})}{\rho_{\sigma, \mathbf{c}}(\Lambda)}$ where $\rho_{\sigma, \mathbf{c}}(\Lambda) = \sum_{\mathbf{x} \in \Lambda} \rho_{\sigma, \mathbf{c}}(\mathbf{x})$.

Tailcut. To tailcut less than $2^{-\lambda}$ of a one-dimensional Gaussian, we use the fact that $\Pr_{\mathbf{x} \leftarrow D_{\mathbb{Z}, \sigma}}[|\mathbf{x}| > t\sigma] \leq \text{erfc}(t/\sqrt{2})$, where $\text{erfc}(x) = 1 - \frac{2}{\pi} \int_0^x \exp^{-t^2} dt$. For example, $t = 12$ for $\lambda = 100$. Then, a vector \mathbf{x} sampled in $D_{\mathbb{Z}^m, \sigma}$ would have small norm $\|\mathbf{x}\| \leq t\sigma\sqrt{m}$ with overwhelming probability.

2.1 Cryptographic Problems on Lattices

Ring-SIS/Ring-LWE. We use ring variants of SIS and LWE, proposed by [LM06, PR06] and [SSTX09, LPR10], and proven to be at least as hard as the GapSVP/SIVP problems on ideal lattices.

Definition 1 (Ring-SIS_{q,m,β}). Given $\mathbf{a} = (a_1, \dots, a_m)^T \in R_q^m$ a vector of m uniformly random polynomials, find a non-zero vector of small polynomials $\mathbf{x} = (x_1, \dots, x_m)^T \in R^m$ such that $\mathbf{a}^T \mathbf{x} = \sum_{i=1}^m a_i \cdot x_i = 0 \pmod q$ and $0 < \|\mathbf{x}\| \leq \beta$.

Definition 2 (Decision Ring-LWE_{n,q,D_{R,σ}}). Given $\mathbf{a} = (a_1, \dots, a_m)^T \in R_q^m$ a vector of m uniformly random polynomials, and $\mathbf{b} = \mathbf{a}s + \mathbf{e}$, where $s \leftarrow U(R_q)$ and $\mathbf{e} \leftarrow D_{R^m, \sigma}$, distinguish $(\mathbf{a}, \mathbf{b} = \mathbf{a}s + \mathbf{e})$ from (\mathbf{a}, \mathbf{b}) drawn from the uniform distribution over $R_q^m \times R_q^m$.

2.2 Dual-Regev Public Key Encryption

The Dual-Regev PKE, first described in [GPV08, Sect. 7.1], is the starting point to build an IBE on lattices. We describe its ring variant, with parameters n, m , and q integers and ζ, τ two real numbers.

- **Key Generation:** The secret key corresponds to a vector of small norm polynomials $\mathbf{x} \in R^m$, sampled from $D_{R^m, \zeta}$. The public key contains a uniformly random chosen vector of polynomials $\mathbf{a} \leftarrow U(R_q^m)$ and one more polynomial $u = \mathbf{a}^T \mathbf{x} \in R_q$.
- **Encryption:** The plaintext message consists in a binary polynomial $M \in R_2$. The ciphertext is composed of $m + 1$ Ring-LWE samples: $(\mathbf{b} = \mathbf{a}s + \mathbf{e}, c = u \cdot s + e' + \lfloor q/2 \rfloor M) \in R_q^m \times R_q$, where s is a uniformly chosen polynomial in R_q , and e', \mathbf{e} follow a discrete Gaussian distribution respectively on R and R^m of parameter τ . The ciphertext is then (\mathbf{b}, c) .
- **Decryption:** The recipient uses his private key \mathbf{x} to compute: $\mu = c - \mathbf{b}^T \mathbf{x} = e' - \mathbf{e}^T \mathbf{x} + \lfloor q/2 \rfloor M$. To recover M , he looks at each coefficient of μ , if μ_i is closer to 0 than to $\lfloor q/2 \rfloor$, the message bit $M_i = 0$, otherwise $M_i = 1$.

Security. The ring-variant of the Dual-Regev scheme is IND-CPA secure under the hardness of Ring-LWE_{n,q,D_{R,τ}} [LPR13]. The correctness of the decryption holds if the error term $\|e' - \mathbf{e}^T \mathbf{x}\|$ is small enough, less than $\lfloor q/4 \rfloor$.

2.3 Cryptographic Definition of an IBE Scheme

Identity Based Encryption. Let λ be the security parameter of the scheme, an Identity Based Encryption (IBE) is composed of four probabilistic polynomial time algorithms (Setup, Extract, Encrypt, Decrypt).

Security Game. We describe the ciphertext indistinguishability under a selective-identity chosen-plaintext attack (IND-sID-CPA) as a game between an adversary \mathcal{A} and a challenger. The game proceeds as follows:

Init: The adversary \mathcal{A} chooses the challenge identity id^* .

Setup: The challenger runs $\text{Setup}(1^\lambda)$, gives the master public key msk to \mathcal{A} .

Queries 1: The adversary can make private-key extraction queries on identities $id \neq id^*$, and the challenger answers by running $sk_{id} \leftarrow \text{Extract}(1^\lambda, mpk, msk, id)$.

Challenge: Adversary \mathcal{A} outputs two plaintexts $M_0, M_1 \in \mathcal{M}$. The challenger chooses a random bit $b^* \leftarrow \{0, 1\}$ and, sets the challenge ciphertext to $C^* = \text{Encrypt}(1^\lambda, mpk, id^*, M_{b^*})$. The challenge ciphertext C^* is sent to \mathcal{A} .

Queries 2: Adversary can make additional queries (answered as in Queries 1).

Guess: Eventually, \mathcal{A} outputs a bit b and wins if $b^* = b$.

The advantage of the adversary \mathcal{A} playing the IND-sID-CPA security game above is $\text{Adv}(\mathcal{A})_{\text{IBE}}^{\text{IND-sID-CPA}} = \left| \Pr[b = b^*] - \frac{1}{2} \right|$. An IBE scheme is IND-sID-CPA secure if, for all PPT adversary \mathcal{A} , his advantage $\text{Adv}(\mathcal{A})_{\text{IBE}}^{\text{IND-sID-CPA}}$ is negligible.

2.4 Hash Functions

Our IBE and signature schemes use an encoding hash function $H : \mathbb{Z}_q^n \rightarrow R_q$ to map identities in \mathbb{Z}_q^n to invertible elements in R_q . The security proof requires the map H to satisfy an injectivity property: the difference of two elements has to be invertible in R_q . Such hash functions have been called *encoding with Full-Rank Differences* (FRD) in [ABB10] and they must satisfy the following properties:

1. for all distinct $u, v \in \mathbb{Z}_q^n$, the element $H(u) - H(v) \in R_q$ is invertible; and
2. H is computable in polynomial time (in $n \log q$).

Implementation of FRD Hash Functions. An invertible element of R_q in the NTT domain corresponds to n non-zero integers of bit-size k . We implement our encoding in a naive manner, by generating these n integers using an PRNG with the identity id as a seed. In the literature, Ducas and Micciancio [DM14] chose the ring R_q , with q a power of 3 because in such ring any polynomial of degree less than $n/2$ with coefficients in $\{-1, 0, 1\}$ is invertible. Recently, Lyubashevsky and Seiler have proposed in [LS18] a way to construct such encoding using the fact that small non-zero polynomials are invertible in cyclotomic rings. They also use that $x^n + 1$ splits modulo q , and perform half of the FFT recursion tree, depending on the logarithm of the number of splittings, and at the end of the FFT tree, for small degree polynomials, multiplications have to be performed using naive or Karatsuba algorithm.

3 Trapdoors on Lattices

The construction of our IBE scheme relies on a kind of trapdoors, introduced by Ajtai [Ajt96], and then improved, in particular in [MP12, GM18]. We define the trapdoor function $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{Ax} \pmod q$ which represents the Inhomogeneous SIS problem. The trapdoor consists in a short basis $\mathbf{T}_{\mathbf{A}} \in \mathbb{Z}^{m \times m}$ of the m -dimensional integer lattice: $\Lambda_q^\perp(\mathbf{A}) := \{\mathbf{x} \in \mathbb{Z}^m \text{ such that } \mathbf{Ax} = \mathbf{0} \pmod q\}$. Thanks to the short basis $\mathbf{T}_{\mathbf{A}}$, we sample from a Gaussian distribution with a small parameter σ to obtain short vectors in $\Lambda_q^\perp(\mathbf{A})$. Then, solve the SIS, ISIS or LWE problems.

We use the ring version of a second notion of trapdoors (gadget based trapdoors) introduced by [MP12], and recently improved by [GM18], which are more efficient. In this construction, the matrix \mathbf{A} is constructed by picking the first part uniformly at random, and the second part is almost uniformly random by including a structured gadget, to help the inversion of the SIS problem.

3.1 Gadget-Based Trapdoor Construction

Gadget vector. We use the gadget vector $\mathbf{g} \in R_q^k$ for which the inversion of $f_{\mathbf{g}^T}(\mathbf{z}) = \mathbf{g}^T \mathbf{z} \in R_q$ is easy: it is a vector of constant polynomials, $\mathbf{g} = (1, 2, 4, \dots, 2^{k-1})^T \in R_q^k$ with $k = \lceil \log_2 q \rceil$. The lattice $\Lambda_q^\perp(\mathbf{g}^T)$ has a publicly known basis $\mathbf{B}_q \in R^{k \times k}$, which satisfies $\|\tilde{\mathbf{B}}_q\| \leq \sqrt{5}$.

Trapdoor construction. The trapdoor construction is an almost uniformly random vector of polynomials $\mathbf{a} \in R_q^m$ starting from a uniformly random vector of polynomials $\mathbf{a}' \in R_q^{m-k}$ (Algorithm 3.1.1) that hides the structured vector \mathbf{g} , together with a trapdoor \mathbf{T} that enables its owner to recover this structure when needed.

Definition 3 (g-trapdoor). Let $\mathbf{a} \in R_q^m$ and $\mathbf{g} \in R_q^k$ with $k = \lceil \log_2 q \rceil$ and $m > k$. A **g-trapdoor** for \mathbf{a} consists in a matrix of small polynomials $\mathbf{T} \in R^{(m-k) \times k}$, following a discrete Gaussian distribution of parameter σ , such that $\mathbf{a}^T \begin{pmatrix} \mathbf{T} \\ \mathbf{I}_k \end{pmatrix} = h\mathbf{g}^T$ for some invertible element $h \in R_q$. The polynomial h is called the tag associated to \mathbf{T} . The quality of the trapdoor is measured by its largest singular value $s_1(\mathbf{T})$.

Algorithm 3.1.1. Algorithm TrapGen($q, \sigma, \mathbf{a}', h$)

Input: q the ring modulus, σ a Gaussian parameter. Optional $\mathbf{a}' \in R_q^{m-k}$ and $h \in R_q$.

If no \mathbf{a}, h is given as input, the algorithm chooses $\mathbf{a}' \leftarrow U(R_q^{m-k})$ and $h = 1$.

Output: $\mathbf{a} \in R_q^m$ with its trapdoor $\mathbf{T} \in R^{(m-k) \times k}$, of norm $\|\mathbf{T}\| \leq t\sigma\sqrt{(m-k)n}$ associated to the tag h .

$\mathbf{T} \leftarrow D_{R^{(m-k) \times k}, \sigma}, \mathbf{a} = (\mathbf{a}'^T \mid h\mathbf{g} - \mathbf{a}'^T \mathbf{T})^T,$

return (\mathbf{a}, \mathbf{T}) .

3.2 Preimage Sampling

Peikert Sampler. To find \mathbf{x} such that $f_{\mathbf{a}^T}(\mathbf{x}) = u$, using this trapdoor, the idea is to find a \mathbf{z} satisfying $f_{\mathbf{g}^T}(\mathbf{z}) = h^{-1} \cdot (u - \mathbf{a}^T \mathbf{p})$ and following a discrete Gaussian distribution of parameter α , where \mathbf{p} is a perturbation vector with covariance matrix $\Sigma_{\mathbf{p}} = \zeta^2 \mathbf{I}_m - \alpha^2 \begin{pmatrix} \mathbf{T} \\ \mathbf{I}_k \end{pmatrix} (\mathbf{T}^T \mathbf{I}_k)$. Then, $\mathbf{x} = \mathbf{p} + \begin{pmatrix} \mathbf{T} \\ \mathbf{I}_k \end{pmatrix} \mathbf{z}$, has covariance matrix $\Sigma_{\mathbf{x}} = \Sigma_{\mathbf{p}} + \alpha^2 \begin{pmatrix} \mathbf{T} \\ \mathbf{I}_k \end{pmatrix} (\mathbf{T}^T \mathbf{I}_k) = \zeta^2 \mathbf{I}_m$ and satisfies $\mathbf{a}^T \mathbf{x} = \mathbf{a}^T \mathbf{p} + \mathbf{a}^T \begin{pmatrix} \mathbf{T} \\ \mathbf{I}_k \end{pmatrix} \mathbf{z} = \mathbf{a}^T \mathbf{p} + h \mathbf{g}^T \mathbf{z} = \mathbf{a}^T \mathbf{p} + h \cdot h^{-1} (u - \mathbf{a}^T \mathbf{p}) = u$. This idea is summarized in Algorithm 3.2.1 `SamplePre`.

Algorithm 3.2.1. Algorithm `SamplePre`($\mathbf{T}, \mathbf{a}, h, \zeta, \sigma, \alpha, u$)

Input: $\mathbf{a} \in R_q^m$, with its trapdoor $\mathbf{T} \in R^{(m-k) \times k}$ associated to an invertible tag $h \in R_q$, $u \in R_q$ and ζ, σ and α three Gaussian parameters.

Output: $\mathbf{x} \in R_q^m$ following a discrete Gaussian distribution of parameter ζ satisfying $\mathbf{a}^T \mathbf{x} = u \in R_q$.

$\mathbf{p} \leftarrow \text{SampleP}(q, \zeta, \alpha, \mathbf{T}), v \leftarrow h^{-1} \cdot (u - \mathbf{a}^T \mathbf{p}),$

$\mathbf{z} \leftarrow \text{SamplePolyG}(\sigma, v), \mathbf{x} \leftarrow \mathbf{p} + \begin{pmatrix} \mathbf{T} \\ \mathbf{I}_k \end{pmatrix} \mathbf{z},$

return \mathbf{x} .

It uses the two following algorithms:

- `SamplePolyG`(σ, v) $\rightarrow \mathbf{z}$, takes as input a Gaussian parameter σ and a target $v \in R_q$, outputs $\mathbf{z} \leftarrow D_{A_q^\perp(\mathbf{g}^T), \alpha, v}$, with $\alpha = \sqrt{5}\sigma$,
- `SampleP`($q, \zeta, \alpha, \mathbf{T}$) $\rightarrow \mathbf{p}$, takes as input the ring modulus q , ζ and α two Gaussian parameters and $\mathbf{T} \leftarrow D_{R^{(m-k) \times k}, \sigma}$, outputs $\mathbf{p} \leftarrow D_{R^m, \sqrt{\Sigma_{\mathbf{p}}}}$, where $\Sigma_{\mathbf{p}} = \zeta^2 \mathbf{I}_m - \alpha^2 \begin{pmatrix} \mathbf{T} \\ \mathbf{I}_k \end{pmatrix} (\mathbf{T}^T \mathbf{I}_k)$ with $\zeta > s_1(\mathbf{T})\alpha$.

4 Identity-Based Encryption

Our IBE construction is a ring-version of the selective IBE of [ABB10], adapted to use the gadget-based trapdoor of Micciancio and Peikert [MP12] (also adapted to rings, described in Sect. 3). We use the same encoding with Full-Rank Differences H to map identities to invertible elements in R_q (defined in Sect. 2.4).

The master public key consists in a uniformly random polynomial $u \in R_q$ and a pseudo-random vector of polynomials $\mathbf{a} \in R_q^m$. The master secret key is a \mathbf{g} -trapdoor $\mathbf{T} \in R^{(m-k) \times k}$ for \mathbf{a} with associated tag set to zero: $\mathbf{a} = (\mathbf{a}^T | -\mathbf{a}^T \mathbf{T})^T$.

Then, thanks to \mathbf{a} , we can compute a publicly computable vector associated to an identity id :

$$\mathbf{a}_{id} = \mathbf{a}^T + (\mathbf{0} | H(id)\mathbf{g})^T = (\mathbf{a}^T | H(id)\mathbf{g} - \mathbf{a}^T \mathbf{T})^T.$$

The secret key associated to an identity id is a short vector $\mathbf{x} \in R^m$, computed thanks to the algorithm `SamplePre`, which satisfies $\mathbf{a}_{id}^T \mathbf{x} = u \in R_q$. We also use the Dual-Regev encryption scheme to encrypt (using \mathbf{a}_{id}) and decrypt (using \mathbf{x}).

4.1 Our Construction

The parameters of the scheme are n, m, q, k , and q integers, and $\sigma, \alpha, \gamma, \tau$, and ζ are real numbers, and chosen as described in Sect. 4.2.

1. $\text{Setup}(1^n) \rightarrow (mpk, msk)$:
 - (a) Compute $\mathbf{a} \in R_q^m$ associated to its trapdoor $\mathbf{T} \in R^{(m-k) \times k}$, $(\mathbf{a}, \mathbf{T}) \leftarrow \text{TrapGen}(q, \sigma, h = 0)$, i.e. $\mathbf{a} = (\mathbf{a}'^T \mid -\mathbf{a}'^T \mathbf{T})^T$,
 - (b) Sample a uniformly random polynomial $u \leftarrow U(R_q)$,
 - (c) Output $mpk = (\mathbf{a}, u) \in R_q^{m+1}$ and $msk = \mathbf{T} \in R^{(m-k) \times k}$.
2. $\text{Extract}(mpk = (\mathbf{a}, u), msk = \mathbf{T}, id \in \mathcal{ID}) \rightarrow sk_{id}$:
 - (a) Compute the tag $h_{id} = H(id)$,
 - (b) Compute $\mathbf{a}_{id} = \mathbf{a}^T + (\mathbf{0} \mid h_{id} \mathbf{g})^T = (\mathbf{a}'^T \mid h_{id} \mathbf{g} - \mathbf{a}'^T \mathbf{T})^T$,
 - (c) Sample a short $\mathbf{x} \leftarrow \text{SamplePre}(\mathbf{T}, \mathbf{a}_{id}, h_{id}, \zeta, \sigma, \alpha, u)$, such that $\mathbf{a}_{id}^T \mathbf{x} = u$,
 - (d) Output $sk_{id} = \mathbf{x} \in R^m$.
3. $\text{Encrypt}(mpk = (\mathbf{a}, u), id, M \in R_2) \rightarrow C$:
 - (a) Compute the tag h_{id} , and \mathbf{a}_{id} like above,
 - (b) Sample $s \leftarrow U(R_q)$, $\mathbf{e}_0 \leftarrow D_{R^{m-k}, \tau}$, $\mathbf{e}_1 \leftarrow D_{R^k, \gamma}$, and $e' \leftarrow D_{R, \tau}$,
 - (c) Compute $\mathbf{b} = \mathbf{a}_{id} s + (\mathbf{e}_0^T \mid \mathbf{e}_1^T)^T \in R_q^m$, and $c = u \cdot s + e' + \lfloor q/2 \rfloor M \in R_q$,
 - (d) Output $C = (\mathbf{b}, c) \in R_q^{m+1}$.
4. $\text{Decrypt}(sk_{id} = \mathbf{x}, C = (\mathbf{b}, c)) \rightarrow M$:
 - (a) Compute $\text{res} = c - \mathbf{b}^T \mathbf{x} = e' - (\mathbf{e}_0^T \mid \mathbf{e}_1^T) \mathbf{x} + \lfloor q/2 \rfloor M \in R_q$,
 - (b) For each res_i , if it is closer to $\lfloor q/2 \rfloor$ than to 0, $M_i = 1$, otherwise $M_i = 0$.

Correctness. Let $\mathbf{x} = (\mathbf{x}_0^T \mid \mathbf{x}_1^T)^T$ with $\mathbf{x}_0 \in R_q^{m-k}$ and $\mathbf{x}_1 \in R_q^k$. To decrypt a ciphertext, we need the error term $e' - (\mathbf{e}_0^T \mid \mathbf{e}_1^T) (\mathbf{x}_0^T \mid \mathbf{x}_1^T)^T = e' - \mathbf{e}_0^T \mathbf{x}_0 - \mathbf{e}_1^T \mathbf{x}_1$ to be bounded by $\lfloor q/4 \rfloor$ (see Sect. 4.2 on the choice of the parameters).

Theorem 1. *Our IBE construction with parameters $n, m, q, k, \sigma, \alpha, \gamma, \tau$ and ζ chosen as below is IND-sID-CPA secure in the standard model under the hardness of Ring-LWE $_{n,q,D_{R,\tau}}$.*

Proof. To prove the IND-sID-CPA security of the IBE scheme described above, we use a sequence of games starting from the original IND-sID-CPA game (Sect. 2.3). In the final game, the adversary has no information left about the initial message and hence has advantage zero. To ensure that the adversary has a negligible advantage in winning the IND-sID-CPA game, we show that a probabilistic polynomial time adversary cannot distinguish between games.

Game 0. The original IND-sID-CPA game between an adversary \mathcal{A} and an IND-sID-CPA challenger.

Game 1. In the *Game 0*, the master public key of the scheme is $mpk = (\mathbf{a}, u)$, with \mathbf{a} generated thanks to $\text{TrapGen}(q, \sigma, h = 0)$ with an associated trapdoor \mathbf{T} (i.e. $\mathbf{a} = (\mathbf{a}^T | -\mathbf{a}^T \mathbf{T})^T$), and u is a uniform polynomial in R_q .

In *Game 1*, we change the generation of the public vector \mathbf{a} by adding information about the challenge identity id^* targeted by \mathcal{A} . The public parameter \mathbf{a} is now generated thanks to $\text{TrapGen}(q, \sigma, \mathbf{a}', -h_{id^*})$, i.e. $\mathbf{a} = (\mathbf{a}^T | -h_{id^*} \mathbf{g} - \mathbf{a}'^T \mathbf{T})^T$, where the first part $\mathbf{a}' \in R_q^{m-k}$ is chosen from the uniform distribution. For the second part, $\mathbf{a}'^T \mathbf{T} = \left(\sum_{i=1}^{m-k} a_i t_{i,1}, \dots, \sum_{i=1}^{m-k} a_i t_{i,k} \right)$, is either computationally or statistically indistinguishable from the uniform distribution depending on the trapdoor instantiation we choose. For the computational instantiation, we set $m - k = 2$, and $\mathbf{a}' = (1, a)$ with $a \leftarrow U(R_q)$ and obtain a public vector $\mathbf{a} = (1, a | -(a \cdot t_{2,1} + t_{1,1}), \dots, -(a \cdot t_{2,k} + t_{1,k}))$. To ensure such \mathbf{a} looks uniform, we use the Ring-LWE assumption in its normal form: where the secret and the error follow the same distribution.

The adversary \mathcal{A} issues private key queries on identities $id \neq id^*$, and the challenger has to answer to these queries. To do that, he computes $\mathbf{a}_{id} = \mathbf{a}^T + (\mathbf{0} | h_{id} \mathbf{g})^T = (\mathbf{a}^T | (h_{id} - h_{id^*}) \mathbf{g} - \mathbf{a}'^T \mathbf{T})^T$, and use $\text{SamplePre}(\mathbf{T}, \mathbf{a}_{id}, h_{id} - h_{id^*}, \zeta, \sigma, \alpha, u)$ to compute a private key associated to id : $\mathbf{x} \in R^m$ satisfying $\mathbf{a}_{id}^T \mathbf{x} = u$, because $h_{id} - h_{id^*}$ is invertible. Note that for $id = id^*$, $\mathbf{a}_{id} = (\mathbf{a}^T | -\mathbf{a}'^T \mathbf{T})^T$, and \mathcal{B} can no longer answer private key queries.

Game 2. In the last game, we change how the challenge ciphertext is build. The ciphertext C^* is now chosen uniformly in $R_q^m \times R_q$. The last step is to show that *Game 1* and *Game 2* are computationally indistinguishable for \mathcal{A} by doing a reduction from the Ring-LWE problem. Suppose that \mathcal{A} has non-negligible advantage in distinguishing these two games. Then a simulator \mathcal{B} can use \mathcal{A} to solve the Ring-LWE problem with non-negligible advantage (Sect. 2.1).

The simulator \mathcal{B} receives $m - k + 1$ samples $(a_i, b_i)_{0 \leq i \leq m-k}$ as an instance of the decisional Ring-LWE problem. The simulator also receives the challenge identity id^* from the adversary \mathcal{A} . Let $\mathbf{a}' = (a_1, \dots, a_{m-k})^T \in R_q^{m-k}$ and $\mathbf{b}' = (b_1, \dots, b_{m-k})^T \in R_q^m$. The simulator runs $\text{TrapGen}(q, \sigma, \mathbf{a}', -h_{id^*})$, because \mathbf{a}' is following a uniform distribution thanks to the Ring-LWE assumption, and gets back $\mathbf{a} = (\mathbf{a}^T | -h_{id^*} \mathbf{g} - \mathbf{a}'^T \mathbf{T})^T$, as in *Game 1*. Next, \mathcal{B} sets $u = a_0$ and sends (\mathbf{a}, u) to \mathcal{A} as the master public key of the scheme. The adversary \mathcal{A} issues private key queries, and \mathcal{B} answer to these queries like in *Game 1*.

Then the attacker \mathcal{A} sends two messages M_0, M_1 to \mathcal{B} . \mathcal{B} generates a random bit b^* , and generates the challenge ciphertext $C^* = (\mathbf{b}^*, c^*)$ as follows: $\mathbf{b}^* = (\mathbf{b}'^T | -\mathbf{b}'^T \mathbf{T} + \hat{\mathbf{e}}^T)^T$, $c^* = b_0 + \lfloor q/2 \rfloor M_{b^*}$, where $\hat{\mathbf{e}} \leftarrow D_{R^k, \mu}$ for some μ real.

- If the Ring-LWE samples are drawn from the Ring-LWE distribution, we have $\mathbf{b}' = \mathbf{a}'s + \mathbf{e}$ and $b_0 = a_0 \cdot s + e_0$, for some $s \in R_q$, $\mathbf{e} \leftarrow D_{R^{m-k}, \tau}$ and $e_0 \leftarrow D_{R, \tau}$. By substitution, $\mathbf{b}^* = (\mathbf{b}'^T | -\mathbf{b}'^T \mathbf{T} + \hat{\mathbf{e}}^T)^T = \mathbf{a}'_{id^*}^T s + (\hat{\mathbf{e}}^T | -\mathbf{e}^T \mathbf{T} + \hat{\mathbf{e}}^T)^T$ and $c^* = b_0 + \lfloor q/2 \rfloor M_{b^*} = a_0 \cdot s + e_0 + \lfloor q/2 \rfloor M_{b^*}$. For fixed \mathbf{e} , the error term $-\mathbf{e}^T \mathbf{T} + \hat{\mathbf{e}}^T$ is indistinguishable from a sample drawn

from the distribution $D_{R^k, \gamma}$ with $\gamma^2 = (\sigma \|\mathbf{e}_0\|)^2 + \mu^2$, for μ well chosen. Then the challenge ciphertext, (\mathbf{b}^*, c^*) follows the same distribution as in the IBE of *Game 2*.

- If the Ring-LWE samples are uniformly random samples, the ciphertext challenge also looks uniform. Then, the challenge ciphertext C^* is always a uniform element in $R_q^m \times R_q$. At the end, the adversary \mathcal{A} outputs a guess b .

If $b = b^*$ with overwhelming probability, the simulator concludes that the Ring-LWE instance was drawn from the Ring-LWE distribution, otherwise \mathcal{B} concludes that the Ring-LWE distribution was drawn from the uniform distribution. \square

4.2 Parameter Choices

In our construction, the modulus q is chosen to be a prime of size 14, 30 or 62 bits (implementation purpose of NTLlib [AMBG+16]). We use the computational instantiation of the trapdoor with $m - k = 2$, and \mathbf{a}' corresponds to two polynomials, the first sets to one and the second to a uniform polynomial $a \leftarrow U(R_q)$, as suggested in [EBB13]. We obtain a public vector, associated to tag h : $\mathbf{a} = (1, a \mid h \cdot g_1 - (a \cdot t_{2,1} + t_{1,1}), \dots, h \cdot g_k - (a \cdot t_{2,k} + t_{1,k}))$.

- The Gaussian parameter σ for the trapdoor sampling is $\sigma > \sqrt{(\ln(2n/\epsilon)/\pi)}$ [MP12] where n is the maximum length of the ring polynomials, and ϵ is the desired bound on the statistical error introduced by each randomized-rounding operation. This parameter is also chosen to ensure that the Ring-LWE instance of parameters n, q and σ is hard.
- The Gaussian parameter α used for the G-sampling [MP12] is $\alpha = \sqrt{5}\sigma$.
- By [GM18], the parameter ζ satisfies $\zeta > s_1(\mathbf{T})\alpha$. The spectral norm of \mathbf{T} is a subgaussian random matrix of parameter σ [MP12, Lemma 2.9]. There exists a universal constant C ($\sim 1/\sqrt{2\pi}$), such that for any $t' \geq 0$, we have $s_1(\mathbf{T}) \leq C\sigma(\sqrt{kn} + \sqrt{2n} + t')$ except with probability at most $2 \exp(-\pi(t')^2)$. Then, we get $\zeta > \sqrt{5}C\sigma^2(\sqrt{kn} + \sqrt{2n} + t')$.
- Finally, to choose the Gaussian parameters γ, τ for the Dual-Regev encryption, we need $\|e' - \mathbf{e}_0^T \mathbf{x}_0 - \mathbf{e}_1^T \mathbf{x}_1\| < \lfloor q/4 \rfloor$, which thanks to Sect. 2 gives:

$$\|e' - \mathbf{e}_0^T \mathbf{x}_0 - \mathbf{e}_1^T \mathbf{x}_1\| \leq t\tau\sqrt{n} + 2t^2\tau\zeta n + t^2\gamma\zeta kn < \lfloor q/4 \rfloor.$$

- Moreover, γ needs to satisfy $\gamma = \sqrt{\sigma^2 \|\mathbf{e}_0\|^2 + \mu^2}$ so that the security proof holds. We can have an idea of the norm of \mathbf{e}_0 by using the Sect. 2,

$$\gamma^2 = \sigma^2 \|\mathbf{e}_0\|^2 + \mu^2 \leq \sigma^2 (t\tau\sqrt{2n})^2 + \mu^2,$$

we choose $\mu = t\sigma\tau\sqrt{2n}$, and then $\gamma = 2t\sigma\tau\sqrt{n}$.

Parameters set. We combine all the conditions to obtain the following set of parameters, used in our implementation. *Remark: The bit security of the underlying Ring-SIS instance does not appear here because it dominates the bit security of the Ring-LWE instances.*

Table 3. Parameters set for our IBE construction.

n	k	σ	LWE $_{\sigma}$	ζ	γ	τ	LWE $_{\tau}$	λ
512	50	3.3	2^{41}	1935.7	4493.2	3.3	2^{41}	40
1024	51	5	2^{80}	6360.5	14747.7	5	2^{80}	80
2048	62	6.7	2^{197}	16898.5	64541.9	6.7	2^{198}	195

4.3 Underlying Signature Scheme

Behind the IBE scheme above, there is an underlying natural signature scheme. The key generation in both scheme consists in creating a public vector $\mathbf{a} \in R_q^m$ together with its trapdoor $\mathbf{T} \in R^{(m-k) \times k}$. The signature of a message corresponds to the secret key associated to an identity in the IBE scheme. A signature \mathbf{x} of a message M is a solution of $\mathbf{a}_M^T \mathbf{x} = 0 \pmod q$. To check if \mathbf{x} is a valid signature for some message M , we have to check that \mathbf{x} is a non trivial solution to the Ring-SIS instance above ($\mathbf{x} \neq \mathbf{0}$, $\mathbf{a}_M^T \mathbf{x} = 0 \pmod q$ and $\|\mathbf{x}\| \leq t\zeta\sqrt{nm}$).

Construction. The parameters of the scheme are n, m, q, k , and q integers, and σ, α , and ζ are reals. Let $H : \mathbb{Z}_q^n \rightarrow R_q$ be a FRD map.

1. **KeyGen**(1^n) $\rightarrow (vk, sk)$:
 - (a) Compute $\mathbf{a} \in R_q^m$ associated to its trapdoor $\mathbf{T} \in R^{(m-k) \times k}$, $(\mathbf{a}, \mathbf{T}) \leftarrow \text{TrapGen}(q, \sigma, h = 0)$, i.e. $\mathbf{a} = (\mathbf{a}^T \mid -\mathbf{a}^T \mathbf{T})^T$,
 - (b) Output $vk = \mathbf{a} \in R_q^m$ and $sk = (\mathbf{a}, \mathbf{T}) \in R_q^m \times R^{(m-k) \times k}$.
2. **Sign**($sk = (\mathbf{a}, \mathbf{T}), M \in R_2$) $\rightarrow \nu$:
 - (a) Compute the tag $h_M = H(M)$,
 - (b) Compute $\mathbf{a}_M = \mathbf{a}^T + (\mathbf{0} \mid h_M \mathbf{g})^T = (\mathbf{a}^T \mid h_M \mathbf{g} - \mathbf{a}^T \mathbf{T})^T$,
 - (c) Sample a short $\mathbf{x} \leftarrow \text{SamplePre}(\mathbf{T}, \mathbf{a}_M, h_M, \zeta, \sigma, \alpha, 0)$, with $\mathbf{a}_M^T \mathbf{x} = 0$,
 - (d) Output $\nu = \mathbf{x} \in R^m$.
3. **Verify**($vk = \mathbf{a}, M, \nu = \mathbf{x}$) $\rightarrow \{\text{accept, reject}\}$:
 - (a) Compute the tag h_M and \mathbf{a}_M like above,
 - (b) Accept if, and only if, $\mathbf{a}_M^T \mathbf{x} = 0 \pmod q$ and $0 < \|\mathbf{x}\| \leq t\zeta\sqrt{mn}$.

Correctness. Thanks to Sect. 2, with high probability the norm of a signature outputted by **SamplePre** is bounded by $t\zeta\sqrt{nm}$, because it is an integer vector of size nm and of Gaussian parameter ζ .

Theorem 2. *Our signature construction with parameters $n, m, q, k, \sigma, \alpha$, and ζ chosen as below is SU-CMA (Selective Unforgeability against Chosen Message Attack) secure in the standard model under the hardness of Ring-SIS $_{q, m-k, \beta}$ with $\beta = (1 + t\sigma\sqrt{(m-k)n})t\zeta\sqrt{mn}$.*

Proof. Let \mathcal{A} be an adversary attacking the signature scheme above through an SU-CMA attack. We build a simulator \mathcal{B} attacking the Ring-SIS problem.

Init. The simulator \mathcal{B} receives $m - k$ uniformly random and independent samples from R_q , $\mathbf{a}' = (a_1, \dots, a_{m-k})^T \in R_q^{m-k}$ as an Ring-SIS instance, and the challenge message M^* from the adversary \mathcal{A} . The simulator runs $\text{TrapGen}(q, \sigma, \mathbf{a}', -h_{M^*})$, and gets back $\mathbf{a} = (\mathbf{a}'^T | -h_{M^*} \mathbf{g} - \mathbf{a}'^T \mathbf{T})^T$ together with $\mathbf{T} \leftarrow D_{R^{(m-k) \times k}, \sigma}$. Next, \mathcal{B} sends $vk = \mathbf{a} \in R_q^m$ to \mathcal{A} .

Signing queries. The adversary \mathcal{A} issues signing queries on messages $M \neq M^*$, and \mathcal{B} has to answer to these queries. To do that, he computes $\mathbf{a}_M = \mathbf{a}^T + (\mathbf{0} | h_M \mathbf{g})^T = (\mathbf{a}'^T | (h_M - h_{M^*}) \mathbf{g} - \mathbf{a}'^T \mathbf{T})^T$, and then he can use $\text{SamplePre}(\mathbf{T}, \mathbf{a}_M, h_M - h_{M^*}, \zeta, \sigma, \alpha, 0)$ to find a signature $\mathbf{x} \in R^m$ satisfying $\mathbf{a}_M^T \mathbf{x} = 0 \pmod q$.

Forgery. Eventually, \mathcal{A} outputs a forgery ν^* for M^* , satisfying $\mathbf{a}_M^T \nu^* = 0 \pmod q$, which gives $\mathbf{a}'^T \underbrace{(I_{m-k} | -\mathbf{T})}_{\mathbf{z}} \nu^* = 0 \pmod q$.

The norm of $\|\mathbf{T}\| \leq t\sigma \sqrt{(m-k)n}$ because each of its columns is a Gaussian vector of size k and of parameter σ . Then, the vector \mathbf{z} is a solution of the Ring-SIS instance of norm $\|\mathbf{z}\| = \|(I_{m-k} | -\mathbf{T}) \nu^*\| \leq \beta = (1 + t\sigma \sqrt{(m-k)n}) \cdot t\zeta \sqrt{mn}$. \square

Parameters choices. The parameters $n, m, q, k, q, \sigma, \alpha$, and ζ follow the same conditions detailed above for the IBE scheme. Moreover, we need to look at the underlying Ring-SIS instance of size $m - k = 2$ and of norm $\beta = (1 + t\sigma \sqrt{2n}) t\zeta \sqrt{mn}$ which corresponds to a SIS instance of size n times bigger. The two following conditions $\beta \geq \sqrt{2n} q^{n/2n} = \sqrt{2n} q$ and $q \geq \beta \sqrt{n} \omega(\log n)$ ensure that the SIS problem has a solution and that is hard.

To get an idea of the security achieved by a SIS instance, we follow the general framework of [APS15, CN11]. To ensure that the shortest vector outputted by BKZ is a solution of our SIS instance of norm $\beta = (1 + t\sigma \sqrt{2n}) t\zeta \sqrt{mn}$, the root Hermite factor δ need to satisfy $\frac{\beta}{q^{n/2n}} = \frac{\beta}{\sqrt{q}} = \delta^{2n}$. To achieve this Root Hermite factor, we need to run BKZ with block size at least b . The estimated cost of running BKZ with block size b , a number of rounds of $N = \frac{(2n)^2}{b^2} \log(2n)$ and on dimension $2n$ is $\text{cost}(\text{BKZ}_{b,N}) \approx \frac{(2n)^3}{b^2} \log(2n) \cdot \text{cost}(\text{SVP oracle})$.

Parameters set. We now combine all those conditions to obtain the following set of parameters. *Remark: We can also instantiate this scheme such that the public key is statistically close to uniform by using a Regularity Lemma ([SS11, Theorem 3.1]), but in this case, $m - k$ is slightly larger and σ is much larger.*

Table 4. Parameters set for our signature scheme.

n	k	σ	LWE_σ	ζ	δ	b	SIS	λ
512	30	4.2	2^{64}	2529.3	1.011380	62	2^{74}	60
1024	24	5.8	2^{378}	6143.8	1.008012	132	2^{156}	140
1024	30	6.3	2^{246}	8023.6	1.007348	154	2^{184}	170

5 Implementation

5.1 General Description

Our implementation was carried out in plain C++11 as a general-purpose library. We now discuss the different principles followed during the design of our code:

Cutting-edge compiler. We have always used the latest version available of the GCC compiler to build our binary. Our final timings have been produced using the GCC 7.2 compiler that allows us to perform various optimization and code sanitization that do not exist in older versions.

Thread-Safety. Lattice-based constructions are known to be highly parallelizable. Therefore, we build our library so that it could be simultaneously called from concurrent threads. To this end, we only use the `<thread>` model of C++11, and not OpenMP or OpenCL, in order to keep the design as simple as possible.

Portability. Modern processors often come with a combination of advanced hardware instructions: SSE 4.1, AVX, AVX2, AVX512+flavors and NEON. These instructions allow developers to boost the performance of their applications. Many developers tend to explicitly include such instructions inside their code to gain in efficiency. However, we do not use this method, since it limits the portability of the code. Instead, we rely on the compiler to insert them depending on the required optimization level and the targeted machine using its own auto-vectorization techniques. Thus, our code can be easily compiled for INTEL and ARM processors without any modification.

Dedicated polynomial ring library. We avoid the use of *generic* number theory libraries, such as NTL and FLINT, in our implementation. Instead, we preferred to use NTLlib library [AMBG+16] which offers fast implementations of arithmetic operations over the ring. However, NTLlib has a primary drawback: it does not allow developers to natively choose a prime modulus q of size between 30 bits and 62 bits. This has resulted in performance penalty on our implementation when q can be of size 50 bits, since we had to use 62 bits.

Double-precision float. Our implementation does not depend on multi-precision floating-point arithmetic. Instead, all floating-point computations are performed using double-precision arithmetic, as in [GM18].

Modularity. We designed our code to be composed of three software layers. In what follows, we describe these layers and discuss some technical choices concerning our implementation.

5.2 Software Layers

Gaussian Preimage Sampling. This layer implements the Peikert Gaussian sampler. As mentioned in Sect. 3.2, this consists of combining two stages: an off-line

(target independent) stage `SampleP` generating perturbation vectors, and an on-line (target dependent) stage `SamplePolyG` generating samples from a particular lattice. We note that this layer does not regard the actual implementation of `SampleP` and `SamplePolyG`. We also note that we never include the runtime execution of `SampleP` in our timings for any operation that does need preimage sampling. Indeed, we suppose that the trusted party, or the signer, periodically calls `SampleP` and stores the resulted outputs. Below, we provide more details about the runtime of this off-line operation.

SampleP and SamplePolyG. This layer implements the recent techniques described in [GM18] and [DP16] for Gaussian sampling. We note that `SamplePolyG` just calls `SampleG` (Fig. 2 in [GM18]) n times to build k polynomials in R_q . For instance, when $n = 1024$ and $k = 30$ bits, `SamplePolyG` calls the underlying sampler 1024 times, and then builds 30 polynomials in their NTT. This n times sampling constitutes the main bottleneck of our Extract/Sign algorithms. As for `SampleP`, we use the Fourier representation in the finite field as well as the FFT's butterfly transformation to speed up all the required multiplication/inversion. To this end, we implement our C++ Cooley-Tukey FFT and optimize it using template class recursion (a template class that recursively uses its own definition).

IBE and Signature. Here, we implement the described signature and IBE schemes using mainly our Gaussian Preimage Sampling and NTLlib to perform arithmetic operations. For the IBE scheme, we build two classes in order to simulate the different roles: the trusted party that generates the master keys and capable of extracting users private keys, and a user that is able to encrypt using the identity of another user as well as to decrypt using its own private key. This abstraction allows us to easily set up our test benchmark with one trusted party and several users. We note that we do not include the timings of the instantiation of the different objects, since it is done only once during the setup of the entire environment, and more importantly, it concerns memory allocations and some initial computations that could be performed otherwise.

5.3 Experimental Results

Our timings have been obtained on an Intel i7-5600 2.6 GHz CPU. Clock cycles were measured with the `high_resolution_clock` class of C++11. Results are provided in Table 5 (resp. Table 6) for the IBE (resp. signature) scheme.

We underline that we obtained our results using the security parameters in Tables 3 and 4 except for k . Indeed, NTLlib limits the choice of k , then we use $k = 30$ if it is smaller or equal to 30, and $k = 62$ otherwise. We note that modifying NTLlib to take into account arbitrary moduli is possible, but here, we did not and thus provide upper bounds to our implementations. This explains our similar results when signing for two different levels of security (with $n = 1024$). Our timings show that performance is still practical for many scenarios. We also notice that timings are almost just multiplied by 2 for twice security.

Table 5. Timings in ms for the different operations of the IBE scheme: Setup, Pre-Compute, Extract, Encrypt and Decrypt.

(λ, n)	Setup	PreCompute	Extract	Encrypt	Decrypt
(40, 512)	0.93	1.32	2.27	0.45	0.0625
(80, 1024)	1.67	3.125	4.02	1.0	0.12
(195, 2048)	3.125	6.67	8.19	2.44	0.94

Table 6. Timings in ms for the different operations of the Signature scheme: KeyGen, PreCompute, Sign and Verify.

(λ, n)	KeyGen	PreCompute	Sign	Verify
(60, 512)	0.52	1.05	1.12	0.025
(140, 1024)	0.91	3.44	2.0	0.043
(170, 1024)	0.96	3.92	1.85	0.047

Comparison with related work. Now, we provide more insight about the Tables 1 and 2 in the introduction, as well as Table 7 presented below.

Table 7. Timings for the different proposals of the NIST competition. Refer to Table 2 for the notations.

Scheme	(λ, n)	KeyGen (ms)	Sign (op/s)	Verify (op/s)
Dilithium [DLL+17]	(128, 768 ^a)	0.08	2263	10660
qTesla [BAA+18]	(128, 1024)	0.88	1267	5938
Falcon [FHK+18]	(195, 768) ^b	53.48	202	2685
DRS [PSDS18]	(128, 1024)	380	50	6
This paper	(140, 1024)	0.91	498	23000

^abased on Module-LWE/SIS with a ring of size 256.

^bcorresponds to 172 bits of quantum security.

IBE Scheme. The closest work to our implemented IBE is the one presented in [DLP14] and re-implemented in [MSO17]. Indeed, both requires Gaussian sampling to extract users' private keys, and our encryption/decryption algorithms are very similar. However, DLP14 relies on NTRU lattices, which allow them to tremendously reduce the size of users keys, and therefore the number of the required Gaussian sampling (only one), while we need n calls to `SampleG`. Our implementation is then slower for the Extract and Encrypt operations, nevertheless, due to our efficient implementation of [GM18], the difference is quite small. However, our Setup is much faster. We recall that the trusted party generates the private key for each user only once. Therefore, this operation is less critical than the encryption/decryption.

Signature Scheme. We compare our implementation with the lattice-based signature proposals of the NIST competition¹. We compiled and ran the *Reference Implementation* of each scheme that corresponds to the *NIST Security Level 1*, namely 128 quantum bit security. Because of some run-time errors, we did not include the timing for pqNTRU_{sign}. We are aware of how different these schemes are in their design and their choice of security parameters. We emphasize that our evaluation study is not complete, especially that we did not implement the secure hash function (as discussed in Sect. 2.4) that might slow down our signature scheme. However, our results allow us to consider our performance compared to highly optimized signature schemes. As for signature, our implementation performs three to four times slower than Dilithium and qTesla, while it outperforms DRS. But our verification is much faster than other schemes. This asymmetry could be useful in case signatures are produced by powerful machines (e.g. a server) and to be verified by constrained devices. These results confirm that our approach remains interesting for both security and performance.

Storage requirements. Here, we give an estimation of the storage requirements of our implementation for the different entities. We note that, in the IBE scheme, the trusted party requires the Gaussian Sampler, while the user requires it for the signature scheme. Our estimations are directly based on the different fields of our classes, which reflects the modular structure of our implementation. We note that the precomputed values take much space, and therefore a trade-off between the number of precomputations and the allocated storage must be found. A direct conclusion can be drawn from our Table 8: lattice-based constructions are not yet ready for constrained devices since, for instance when $n = 512$ and $k = 30$, the public key is of size 61.875 KB, which is quite big for some systems.

Table 8. Storage requirements in bits

(a) Trusted Party		(b) Cipher/Signature	
Public Key	$nk^2 + 3nk$	Cipher	$3nk + nk^2$
Private Key	$2nk$	Signature	$2nk + nk^2$
(c) User		(d) Gaussian Sampler	
Public Key	$3nk + nk^2$	SampleP	$384n$
Private Key	$2nk + nk^2$	SampleG	$192n$
		Precomputations	$2kn + nk^2$

¹ We got all the codes from the NIST website <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions>.

Acknowledgments. This work has received a French government support managed by the National Research Agency in the “Investing for the Future” program, under the national project RISQ P141580-2660001/DOS0044216, and under the projet TYREX granted by the CominLabs excellence laboratory with reference ANR-10-LABX-07-01. Pauline Bert is funded by the Direction Générale de l’Armement (Pôle de Recherche CYBER).

References

- [ABB10] Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_28
- [Ajt96] Ajtai, M.: Generating hard instances of lattice problems. In: Proceedings of STOC, pp. 99–108. ACM (1996)
- [AMBG+16] Aguilar-Melchor, C., Barrier, J., Guelton, S., Guinet, A., Killijian, M.-O., Lepoint, T.: NFLIB: NTT-based fast lattice library. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 341–356. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_20
- [AP09] Alwen, J., Peikert, C.: Generating shorter bases for hard random lattices. In: STACS. Citeseer (2009)
- [APS15] Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. *J. Math. Cryptol.* **9**, 169–203 (2015)
- [BAA+18] Bindel, N., Akleyk, S., Alkim, E., Barreto, P., Buchmann, J., Eaton, E., Gutoski, G., Kramer, J., Longa, P., Polat, H., Ricardini, J., Zanon, G.: qTesla, January 2018
- [BF01] 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
- [BGG+14] Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 533–556. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_30
- [CHKP10] Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 523–552. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_27
- [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
- [Coc01] Cocks, C.: An identity based encryption scheme based on quadratic residues. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 360–363. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45325-3_32

- [DDLL13] 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
- [DDP+17] Dai, W., Doröz, Y., Polyakov, Y., Rohloff, K., Sajjadpour, H., Savaş, E., Sunar, B.: Implementation and evaluation of a lattice-based key-policy ABE scheme. Cryptology ePrint Archive, Report 2017/601 (2017)
- [DLL+17] Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehle, D.: CRYSTALS - dilithium: digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633 (2017)
- [DLP14] Ducas, L., Lyubashevsky, V., Prest, T.: Efficient identity-based encryption over NTRU lattices. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 22–41. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_2
- [DM14] Ducas, L., Micciancio, D.: Improved short lattice signatures in the standard model. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 335–352. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_19
- [DP16] Ducas, L., Prest, T.: Fast fourier orthogonalization. In: ISSAC 2016, pp. 191–198. ACM (2016)
- [EBB13] El Bansarkhani, R., Buchmann, J.: Improvement and efficient implementation of a lattice-based signature scheme. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 48–67. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_3
- [FHK+18] Fouque, P., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: fast-Fourier lattice-based compact signatures over NTRU, January 2018
- [Fou13] Fouotsa, E.: Calcul des couplages et arithmetique des courbes elliptiques pour la cryptographie. Ph.D. thesis (2013)
- [GM18] Genise, N., Micciancio, D.: Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In: EUROCRYPT 2018 (2018, in press)
- [GPR+17] Doruk Gür, K., Polyakov, Y., Rohloff, K., Ryan, G.W., Savaş, E.: Implementation and evaluation of improved Gaussian sampling for lattice trapdoors. Cryptology ePrint Archive, Report 2017/285 (2017)
- [GPV08] Gentry, C., Peikert, C., Vaikuntanathan, V.: How to use a short basis: trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of STOC (2008)
- [LM06] Lyubashevsky, V., Micciancio, D.: Generalized compact knapsacks are collision resistant. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 144–155. Springer, Heidelberg (2006). https://doi.org/10.1007/11787006_13
- [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
- [LPR13] 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
- [LS18] Lyubashevsky, V., Seiler, G.: Partially splitting rings for faster lattice-based zero-knowledge proofs. In: EUROCRYPT 2018 (2018, in press)

- [Lyu12] Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43
- [Mic07] Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Comput. Complex.* **16**(4), 365–411 (2007)
- [MP12] Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
- [MSO17] McCarthy, S., Smyth, N., O’Sullivan, E.: A practical implementation of identity-based encryption over NTRU lattices. *Cryptology ePrint Archive*, Report 2017/1049 (2017)
- [PR06] Peikert, C., Rosen, A.: Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 145–166. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_8
- [PSDS18] Plantard, T., Sipasseuth, A., Dumondelle, C., Susilo, W.: Diagonal dominant reduction for lattice-based signature, January 2018
- [Reg05] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: *Proceedings of STOC* (2005)
- [Sha85] Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-39568-7_5
- [Sho97] Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**(5), 1484 (1997)
- [SS11] Stehlé, D., Steinfeld, R.: Making NTRU as secure as worst-case problems over ideal lattices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 27–47. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_4
- [SSTX09] Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 617–635. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_36



Progressive Lattice Sieving

Thijs Laarhoven¹(✉)  and Artur Mariano²

¹ Eindhoven University of Technology, Eindhoven, The Netherlands
mail@thijs.com

² University of Coimbra, Coimbra, Portugal
artur.mariano@co.ip.pt

Abstract. Most algorithms for hard lattice problems are based on the principle of rank reduction: to solve a problem in a d -dimensional lattice, one first solves one or more problem instances in a sublattice of rank $d-1$, and then uses this information to find a solution to the original problem. Existing lattice sieving methods, however, tackle lattice problems such as the shortest vector problem (SVP) directly, and work with the full-rank lattice from the start. Lattice sieving further seems to benefit less from starting with reduced bases than other methods, and finding an approximate solution almost takes as long as finding an exact solution. These properties currently set sieving apart from other methods.

In this work we consider a progressive approach to lattice sieving, where we gradually introduce new basis vectors only when the sieve has stabilized on the previous basis vectors. This leads to improved (heuristic) guarantees on finding approximate shortest vectors, a bigger practical impact of the quality of the basis on the run-time, better memory management, a smoother and more predictable behavior of the algorithm, and significantly faster convergence – compared to traditional approaches, we save between a factor 20 to 40 in the time complexity for SVP.

Keywords: Lattice-based cryptography · Lattice sieving
Shortest vector problem (SVP) · Nearest neighbor searching

1 Introduction

Finding short lattice vectors. A central hard problem in the study of lattices is the shortest vector problem (SVP): given a lattice $\mathcal{L} = \{\sum_{i=1}^d c_i \mathbf{b}_i : c_i \in \mathbb{Z}\} \subset \mathbb{R}^d$, find a non-zero lattice vector \mathbf{s} of minimum norm, i.e. find $\mathbf{s} \in \mathcal{L}$ satisfying $\|\mathbf{s}\| = \lambda_1(\mathcal{L}) := \min_{\mathbf{x} \in \mathcal{L} \setminus \{0\}} \|\mathbf{x}\|$. The security of many lattice-based cryptographic primitives can be traced back to the hardness of SVP or approximate SVP (γ -SVP), where a non-zero vector $\mathbf{s} \in \mathcal{L}$ of norm at most $\gamma \cdot \lambda_1(\mathcal{L})$ suffices as a solution. Being able to estimate the computational hardness of these problems is crucial for accurately assessing capabilities of cryptographic adversaries, and for selecting parameters in cryptographic schemes [MLC+17]. Algorithms for

exact SVP are essential both for solving exact and approximate SVP, as the latter problem can (heuristically) be reduced to solving several exact SVP instances in lower-dimensional lattices through the BKZ algorithm [Sch87, SE94, CN11].

Exact SVP algorithms. There are currently two classes of algorithms for exact SVP: algorithms requiring superexponential time $2^{\omega(d)}$ in the lattice dimension d , using a negligible amount of memory (such as enumeration [Kan83, FP85, SE94, GNR10, AN17]), and algorithms requiring single exponential time and space $2^{\Theta(d)}$ (such as lattice sieving [AKS01, HPS11] and Voronoi cell approaches [SFS09, MV10a, Laa16]). Although the latter methods have clear drawbacks due to the large memory requirement, these algorithms will inevitably surpass the former algorithms in terms of the time complexity in high dimensions. Conservative estimates of the (post-quantum) hardness of SVP are therefore often based on the state-of-the-art asymptotics for the best (quantum) lattice sieving algorithms [ADPS16, DLL+18, BDK+18].

Lattice sieving. Lattice sieving was introduced in 2001 by Ajtai–Kumar–Sivakumar [AKS01], and was only made somewhat practical less than 10 years ago with fast heuristics [NV08, MV10b]. Recent years have seen further developments in decreasing the asymptotic time complexity of sieving at the cost of using more space [WLTB11, ZPH13, BGJ14, Laa15, BGJ15, LdW15, BL16, BDGL16], while tradeoffs in the reverse direction have also been studied recently to reduce the large memory requirement [BLS16, HK17, HKL18]. Various efforts have further been made to make these algorithms competitive in high-performance computing environments [Sch11, MS11, Sch13, FBB+14, MTB14, MODB14, IKMT14, MLB15, BNvdP16, MB16, MLB17, YKYC17, Duc18]. The theoretically fastest method in high dimensions is currently the LDSieve, with asymptotic time and space complexities $2^{0.29d+o(d)}$ [BDGL16], while in practice the GaussSieve and HashSieve appear to be the most practical in high dimensions [MB16, MLB17, YKYC17].

Differences with other approaches. Existing lattice sieving methods are fundamentally different from other SVP approaches in several ways. Whereas for instance enumeration and Voronoi cell-based methods use a rank reduction step to reduce a d -dimensional problem to problems in lattices of rank $d - 1$, lattice sieving never considers other lattices than the full-rank one. And unlike other methods, lattice sieving (i) does not appear to benefit greatly from being given better lattice bases as input (i.e. BKZ-reduced bases with larger block sizes), and (ii) does not appear to be much faster when an approximate solution suffices – often the algorithm only starts to find shorter vectors after the algorithm is already 80% finished with finding an exact solution. All these properties currently set sieving apart from other methods, and so one might ask whether lattice sieving can be adjusted and perhaps improved by applying similar rank-reduction techniques, so that sieving also obtains these “natural” properties of finding short vectors faster, and performing better when given more reduced bases.

1.1 Contributions

We present *progressive lattice sieving* as a new baseline sieving approach, which resolves many of the above differences with other methods, and greatly improves the performance of heuristic sieving algorithms in practice. Progressive lattice sieving uses a bottom-up approach, initially starting with sieving in a low-rank sublattice of the input lattice. Then, when this subspace of the original space has been saturated with vectors from this sublattice, new basis vectors are introduced to find shorter lattice vectors in sublattices of higher rank, until ultimately the algorithm reaches the full-rank lattice and attempts to find short vectors in the original lattice. Using this rank reduction approach, progressive sieving offers various benefits over standard sieving techniques, which we list below. These are mostly independent of the exact underlying sieve used (e.g. the GaussSieve [MV10b], HashSieve [Laa15], or LDSieve [BDGL16]), although in some cases the improvements are bigger than in other cases.

Faster convergence: Overall, progressive sieving finds shortest vectors in lattices much faster than using current methods, with speedups as large as a factor 40 in dimension 70. Tables 1 and 2 illustrate improvements for 70-dimensional lattices for the GaussSieve and HashSieve, and Fig. 1 shows the improvements for various dimensions using progressive sieving.

Better memory usage: Working on low-rank lattices requires fewer vectors to make progress, allowing one to do a large part of the algorithm using much less memory than current approaches. Experiments further show progressive sieving uses slightly less memory overall (Fig. 2b). This is especially relevant as the main bottleneck in sieving is hitting the memory wall [MB16].

Heuristic guarantees for approximate SVP: Using the Geometric Series Assumption, progressive sieving heuristically finds approximate solutions faster than the full solution, unlike other approaches which commonly require a large number of vectors and reductions to make any progress at all.¹

Larger impact of reduced bases: The more reduced the input basis is, the faster shorter lattice vectors are found, which contributes to a faster overall running time. Similar to pruning in enumeration, this further opens up some new directions for potential improvements (see Sect. 5.2).

Better predictability: As illustrated in the experimental profiles (Figs. 2 and 3), various aspects of sieving are easier to predict and easier to explain theoretically with progressive sieving, which may lead to more accurate predictions when extrapolating to higher dimensions.

Less resource contention: Vectors in the sieved list are only modified sporadically. Instead of using the lock-free mechanism of [MB16,MLB17], to avoid collisions between different nodes, we may therefore be able to construct parallel implementations with lower-overhead incurring mechanisms.

¹ Arguably for current sieving approaches one could also take a sublattice of the full lattice, based on the GSA, and do sieving on that lattice. However, in that case the lattice may be too small (no solutions found) or too big (taking too much time). With progressive sieving no a priori choices need to be made (see also Sect. 5.1).

Outline. The remainder of this paper is organized as follows. In Sect. 2, we introduce notation and related work on lattice sieving. Section 3 describes progressive sieving, and an experimental comparison with previous approaches is given in Sect. 4. Section 5 discusses various aspects of these different approaches to sieving, and Sect. 6 concludes with open problems for future work.

Table 1. Running times (in seconds) for solving exact SVP on 70-dimensional lattices from the SVP challenge [SVP18], using the baseline GaussSieve algorithm [MV10b] and the near-neighbor-based HashSieve extension of the GaussSieve [Laa15]. The lattice bases are pre-reduced with either LLL or BKZ with blocksize 10 or 30, where we used `fp111` [fp1118] to perform the basis reduction. The timings above are based only on the sieving step, and do not include the time for the basis reduction step.

Exact SVP	← GaussSieve →			← HashSieve →		
	LLL	BKZ-10	BKZ-30	LLL	BKZ-10	BKZ-30
Standard sieving	19100	18100	16500	3300	3050	2900
Progressive sieving	595	440	390	165	125	115
Speedup factor	32×	41×	42×	20×	24×	25×

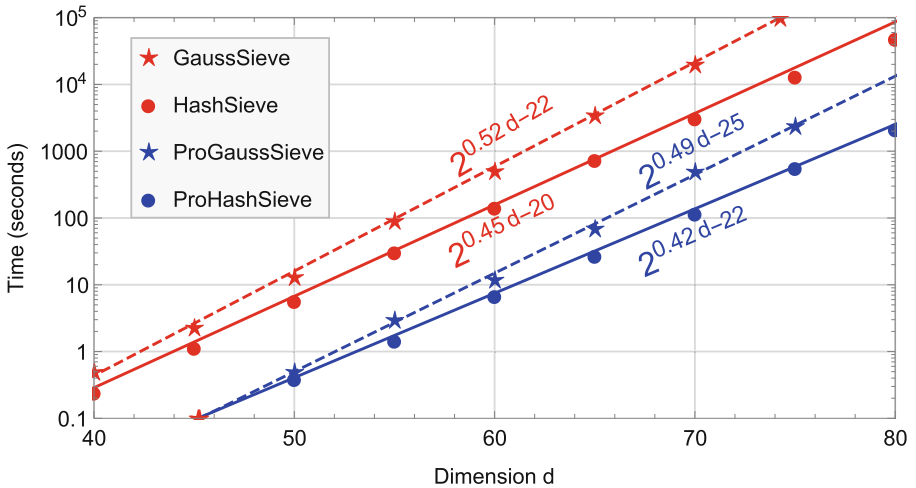


Fig. 1. Time complexities for solving exact SVP on BKZ-10 reduced bases for the SVP challenge lattices, using standard lattice sieving approaches (GaussSieve and HashSieve) and progressive sieving modifications of these algorithms (ProGaussSieve and ProHashSieve). The formulas written in the figure denote the least-squares fits of the four data sets up to two significant digits.

Concurrent work. During the write-up of this paper, we learned similar ideas were independently and concurrently being studied by Ducas, which were later published in [Duc18]. The focus of that work however is on a different improvement

to sieving (“a few dimensions for free”), and the idea of progressive sieving is only mentioned in passing, with little explanation where the improvement comes from. In this paper we chose to focus only on the progressive sieving idea, to understand why it works and where further improvements can be found.

2 Preliminaries

Given a set $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_m\} \subset \mathbb{R}^d$ of m independent d -dimensional vectors, we denote the (rank- m) lattice spanned by \mathbf{B} by $\mathcal{L}(\mathbf{B}) = \{\sum_{i=1}^m c_i \mathbf{b}_i : c_i \in \mathbb{Z}\}$. With abuse of notation, we sometimes write $\mathcal{L} = \mathcal{L}(\mathbf{B})$ when the basis \mathbf{B} is implicit. For $k \leq m$, we write $\mathcal{L}_k = \mathcal{L}_k(\mathbf{B})$ for the sublattice $\{\sum_{i=1}^k c_i \mathbf{b}_i : c_i \in \mathbb{Z}\} \subseteq \mathcal{L}$ spanned by the first k basis vectors of \mathbf{B} . For simplicity, we commonly assume that the original problem concerns a lattice problem in a lattice of full rank ($m = d$); however, using rank reduction we sometimes consider problems in sublattices as well. We denote by $\text{Vol}(\mathcal{L}) = \sqrt{\det(\mathbf{B}^T \mathbf{B})}$ the volume of a lattice \mathcal{L} , and by $\text{Vol}(\mathcal{A})$ the volume of a set $\mathcal{A} \subset \mathbb{R}^d$.

2.1 Heuristic Assumptions

To analyze lattice algorithms, various heuristic assumptions are commonly used. These are not proven and may well be false for certain extreme lattices/bases, but commonly hold for *random* lattices encountered in cryptography. Analyzing algorithms under these (average-case) assumptions therefore commonly leads to more accurate security assessments than analyses based on provable (worst-case) bounds, which are commonly not tight for random lattices.

The *Gaussian heuristic* states that, given a subset $\mathcal{A} \subset \mathbb{R}^d$, the number of lattice points in \mathcal{A} is roughly $|\mathcal{A} \cap \mathcal{L}| \approx \text{Vol}(\mathcal{A}) / \text{Vol}(\mathcal{L})$. As a consequence, we expect a ball of radius r around a random point in space to contain a lattice point iff $r \geq r_0 \sim \sqrt{d/(2\pi e)}$, and heuristically we expect $\lambda_1(\mathcal{L}) \approx \sqrt{d/(2\pi e)}$.

The *Geometric Series Assumption (GSA)* states that, after performing BKZ reduction [Sch87] with block-size β on a full-rank lattice basis, the Gram-Schmidt orthogonalization $\mathbf{b}_1^* \dots \mathbf{b}_d^*$ of the output basis satisfies $\|\mathbf{b}_i^*\| \approx \delta^{d-2i+1} \cdot \text{Vol}(\mathcal{L})^{1/d}$ for all i . Here $\delta = \delta(\beta)$ determines the quality of the basis; the smaller δ , the more orthogonal the basis and the shorter \mathbf{b}_1 . Note that $\prod_{i=1}^d \|\mathbf{b}_i^*\| = \text{Vol}(\mathcal{L})$.

2.2 Lattice Sieving Algorithms

The Nguyen–Vidick sieve. When the idea of lattice sieving was introduced by Ajtai–Kumar–Sivakumar [AKS01], it was still a purely theoretical idea, and no practical instantiations existed until Nguyen–Vidick described a heuristic version of this idea in [NV08]. This sieve starts by sampling many long lattice vectors (e.g. from a discrete Gaussian over the lattice), and then iteratively applies a sieve to this list of vectors to produce a list with fewer, shorter lattice vectors. This sieve essentially works by considering all pairwise combinations of vectors in the input list, and seeing if any of the sums/differences are shorter than the

Algorithm 1. The **standard** GaussSieve algorithm – **GaussSieve**

```

1: Initialize an empty list  $L \leftarrow \emptyset$  and an empty stack  $S \leftarrow \emptyset$ 
2: Initialize COLLISIONS  $\leftarrow 0$ 
3: while true do
4:   Get a vector  $v$  from the stack  $S$  or sample a new one from  $\mathcal{L}(b_1, \dots, b_d)$ 
5:   Reduce  $v$  with all  $w \in L$  // this naively takes  $O(|L|)$  time
6:   Reduce all  $w \in L$  with  $v$  // NNS can speed up these searches
7:   Move reduced vectors  $w \in L$  from the list  $L$  to the stack  $S$ 
8:   if  $v$  has not changed then
9:     Add  $v$  to the list  $L$ 
10:  else
11:    if  $v \neq 0$  then
12:      Add  $v$  to the stack  $S$ 
13:    else
14:      COLLISIONS++
15:      if COLLISIONS = 100 then // a target norm may be used instead
16:        return  $\operatorname{argmin}_{v \in L} \|v\|$  // the shortest vector found so far

```

original vectors. Note that any such combination of lattice vectors again forms a vector which is in the lattice. These pairwise combinations are then used in the next iteration, while the input vectors to the sieve are discarded. This process of throwing away many short lattice vectors is unnaturally wasteful, and the Nguyen–Vidick sieve is therefore not commonly used in practice.

The GaussSieve and ListSieve. In 2010, Micciancio–Voulgaris [MV10b] introduced two new (heuristic) lattice sieving algorithms, which unlike the Nguyen–Vidick sieve both start from an empty list L , and gradually grow this list by adding more (short) lattice vectors to the list, until this list contains a solution.

In the *ListSieve*, before adding a randomly sampled lattice vector v to the list, the vector is reduced with all list vectors $w \in L$ by checking whether either of the sum/difference vectors $v \pm w$ are shorter than v – if so, v is replaced by this shorter vector. Once v has been reduced with all list vectors, and no pairwise sums/differences with list vectors lead to shorter vectors anymore, v is added to the list. This process is repeated until L contains a shortest vector.

In the *GaussSieve* this process is essentially the same, except that list vectors $w \in L$ are also reduced with newly sampled vectors v , before adding v to the list; in contrast, in the *ListSieve* vectors which have been added to the list L are never modified again. Algorithm 1 describes the *GaussSieve*, while removing Lines 6–7 leads to the *ListSieve* algorithm.

Extensions and variants. Over the last few years, various extensions and variants of these heuristic sieving algorithms have been studied, to make them even more efficient in practice. High-dimensional nearest neighbor search algorithms have been used to obtain speed-ups in the searches in Lines 5–6 of Algorithm 1, which naively take time $O(|L|)$ but can be done in sublinear time $O(|L|^\rho)$ with $\rho < 1$ with more advanced methods of indexing and querying the list L [Laa15, LdW15,

[BGJ15, BL16, BDGL16]. Recent work on tuple sieving [BLS16, HK17, HKL18] focused on reducing the memory of lattice sieving, by considering a larger number of vectors $\mathbf{w}_1, \dots, \mathbf{w}_k \in L$ from the list to form short combinations $\mathbf{v} \pm \mathbf{w}_1 \pm \dots \pm \mathbf{w}_k$ with \mathbf{v} . Other work has shown that the same techniques can be used to solve the Closest Vector Problem (CVP), and how near neighbor techniques can be used to obtain fast CVP algorithms with preprocessing [Laa16].

3 Progressive Lattice Sieving

When given a (full-rank) d -dimensional lattice basis, standard sieving approaches attempt to saturate this entire d -dimensional space from the start, always working with vectors from the full-dimensional space. With pairwise reductions between lattice vectors, one needs roughly $2^{0.21d+o(d)}$ vectors to saturate this space and make significant progress in obtaining shorter vectors. This means that also approximate shorter vectors are not found until the list size approaches $2^{0.21d+o(d)}$, which means a lot of effort is spent even before any progress is made at all. Naively this means the time complexity to find any shorter vectors at all is at least $2^{0.42d+o(d)}$ (or $2^{0.29d+o(d)}$ using near neighbor techniques [BDGL16]).

Using BKZ, finding approximate shortest vectors does not necessarily require the same amount of effort as finding exact solutions. More specifically, BKZ only solves SVP instances in lower-dimensional lattices, thereby finding short lattice vectors in (projected) sublattices sooner. The idea of progressive lattice sieving is similar: by first considering low-dimensional sublattices of the full lattice, which already contain many short lattice vectors, we will find approximate solutions faster, which will eventually contribute to finding exact solutions faster as well.

The GaussSieve. Applying progressive sieving to the GaussSieve means making the following modifications. First, we start with a sublattice of the original lattice, spanned by the first few basis vectors. We then run a sieve on this sublattice, until we reach the stopping criterion (e.g. a certain number of collisions). We then add a new basis vector to the search space, and continue sieving in this sublattice of slightly higher rank. We repeat this procedure until we reach the complete lattice, where we expect to find an exact solution to SVP. The result of applying progressive sieving to the GaussSieve is sketched in Algorithm 2, where the main modifications are (1) vectors are sampled from sublattices, and (2) the rank counter is increased and the collision counter is reset when we reach 100 collisions. For simplicity we start with a sublattice of rank 10 – everything below rank 30 or 40 generally finishes within a second anyway, so setting the initial rank to any rank below 40 will lead to similar results in higher dimensions.

Other sieving algorithms. Naturally the same idea can be trivially applied to most other sieving algorithms as well. Applying the same idea to the List-Sieve [MV10b] would simply mean we do not reduce list vectors with sampled vectors in Lines 6–7 of Algorithm 2. Near neighbor extensions to sieving [Laa15, BDGL16] only affect the search procedure for finding reducing pairs of

Algorithm 2. The **progressive** GaussSieve algorithm – **ProGaussSieve**

```

1: Initialize an empty list  $L \leftarrow \emptyset$  and an empty stack  $S \leftarrow \emptyset$ 
2: Initialize COLLISIONS  $\leftarrow 0$ , RANK  $\leftarrow \min\{10, d\}$ 
3: while true do
4:   Get a vector  $v$  from the stack  $S$  or sample a new one from  $\mathcal{L}(b_1, \dots, b_{\text{RANK}})$ 
5:   Reduce  $v$  with all  $w \in L$  // this naively takes  $O(|L|)$  time
6:   Reduce all  $w \in L$  with  $v$  // NNS can speed up these searches
7:   Move reduced vectors  $w \in L$  from the list  $L$  to the stack  $S$ 
8:   if  $v$  has not changed then
9:     Add  $v$  to the list  $L$ 
10:  else
11:    if  $v \neq 0$  then
12:      Add  $v$  to the stack  $S$ 
13:    else
14:      COLLISIONS++
15:      if COLLISIONS = 100 then
16:        if RANK =  $d$  then
17:          return  $\operatorname{argmin}_{v \in L} \|v\|$  // the shortest vector found so far
18:        else
19:          RANK++ // continue with the next sublattice
20:          COLLISIONS  $\leftarrow 0$  // reset the collisions counter

```

vectors, and this can naturally be combined with progressive sieving as well. For tuple sieving [BLS16, HK17, HKL18] we simply run tuple sieving on sublattices until the corresponding sublattice has been saturated, after which we increase the rank as before. Applying progressive sieving to the CVP sieve of [Laa16] is also straightforward. The same idea can also be applied to the Nguyen–Vidick sieve, but in that case the integration of progressive sieving is slightly more contrived; since the practical results with the Nguyen–Vidick sieve will likely be worse than when using the GaussSieve, we have chosen to omit this application.

4 Experiments

Heuristically, sieving naively requires a search over all pairs of vectors in the list (when not using near neighbor techniques), leading to a quadratic time complexity in the list size. With progressive sieving, these arguments still hold, and so the asymptotic improvement of progressive sieving will not be visible in the leading time/memory exponents. Experimentally however there are rather large hidden order terms, and these may become smaller with progressive sieving. To get an insight into this improvement, we will experimentally evaluate progressive sieving and compare it to standard sieving approaches. As a baseline we will use the GaussSieve [MV10b], the baseline sieve without near neighbor searching, and the HashSieve [Laa15], which is perhaps the most practical near neighbor extension of the GaussSieve to date [MLB15, MB16, MLB17]. In Sect. 5 we will give a more in-depth discussion of various aspects of these results.

Experiment setting. All experiments were performed on a Medion Erazer P6661 laptop with an Intel Core i7-6500 CPU (2.50 GHz) and 8 GB of RAM. Except for approximate SVP, where the termination condition was finding a vector of a certain length, all experiments used a bound on the number of “collisions” (reducing to the all-zero vector) as the termination condition. Similar to e.g. [MV10b, IKMT14] we used (an unoptimized version of) Klein’s sampler [Kle00]. For the HashSieve, we used the same parameters as in [Laa15, MLB15]. All experiments were performed on lattices from the SVP challenge [SVP18], where the LLL/BKZ basis reduction was done using `fp111` [fp1118].

4.1 Profiles

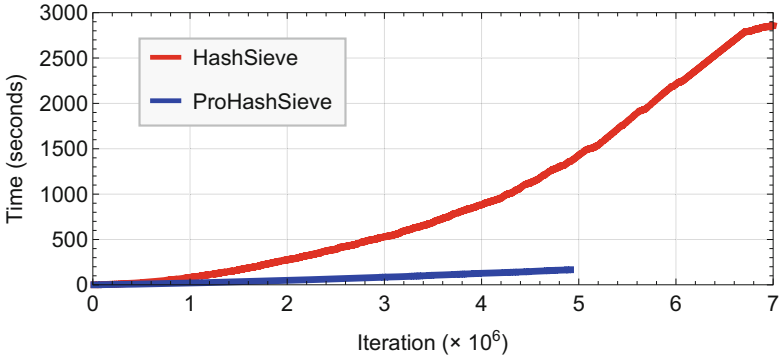
To get an idea how progressive sieving differs from classical sieving in practice, Figs. 2 and 3 depict typical profiles of the classical and progressive HashSieve, when solving SVP on 70-dimensional lattices from the SVP challenge [SVP18]. As described in the captions of the figures, the time complexities are significantly better, the list size increases more steadily (and predictably) than in the original HashSieve, approximate solutions are found considerably faster, and vectors in the list are updated much less frequently in progressive sieving.

Note that as in Fig. 2c, classical sieving can be viewed as a special case of progressive sieving, with the initial value of RANK set to d . Also note that in all figures, the horizontal axis counts the number of *iterations* of the **while**-loop of Algorithms 1 and 2), which translates to actual time complexities as in Fig. 2a – this means that e.g. a vector of Euclidean norm 2400 in this 70-dimensional lattice is not found a factor $5\times$ faster (as one might guess from Fig. 3a), but close to a factor $5 \cdot 20 \approx 1000\times$ faster (taking into account Fig. 2a).

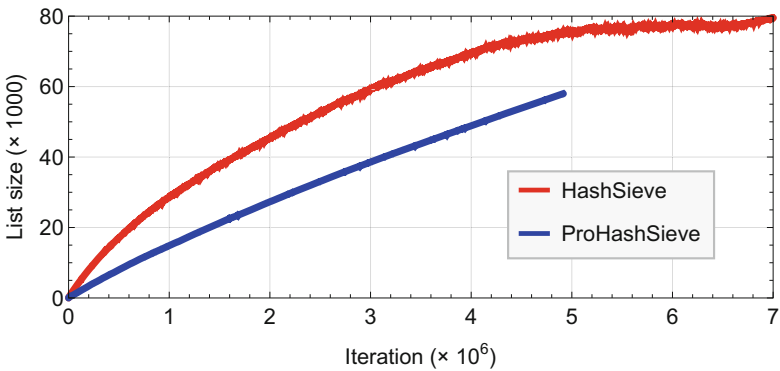
4.2 Results

To get reliable complexity estimates for arbitrary dimensions, we ran several experiments of both standard and progressive sieving on lattices of dimensions 40 to 80, the results of which are displayed in Fig. 1. These results clearly show a large decrease in the time complexities for sieving in arbitrary dimensions, both for the classical GaussSieve and the near-neighbor-optimized HashSieve algorithm [Laa15]. For both algorithms the improvement is more than a factor $20\times$ for all tested dimensions, showing that the improvement of progressive sieving can indeed be combined with near neighbor techniques.

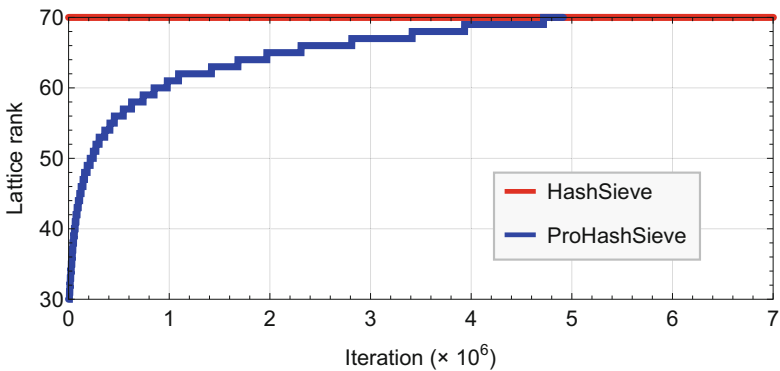
Focusing only on 70-dimensional lattices, we also ran experiments with different quality bases, to see how the basis reduction affects the performance of (progressive) sieving. The results in Table 1 are based on at least 10 experiments each on randomized lattice bases (where only the experiments that took several hours are based on just 10 runs). As can be seen in the table, progressive sieving benefits more from reduced bases, with the speedup factor increasing as the quality of the basis improves. We further observe that the speedup factor for the HashSieve is slightly smaller than for the GaussSieve, which can be explained by the HashSieve spending most of its time working with the near neighbor data structure – costs which are not affected by progressive sieving.



(a) **Time complexities.** Iterations (Line 3) take significantly less time in the ProHashSieve, leading to finding short lattice vectors much faster.

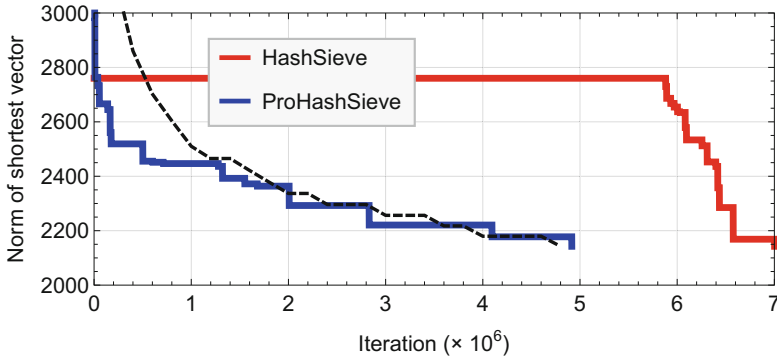


(b) **List sizes.** Whereas the list size behaves rather erratically in the original HashSieve, the ProHashSieve shows a much more steady behavior.

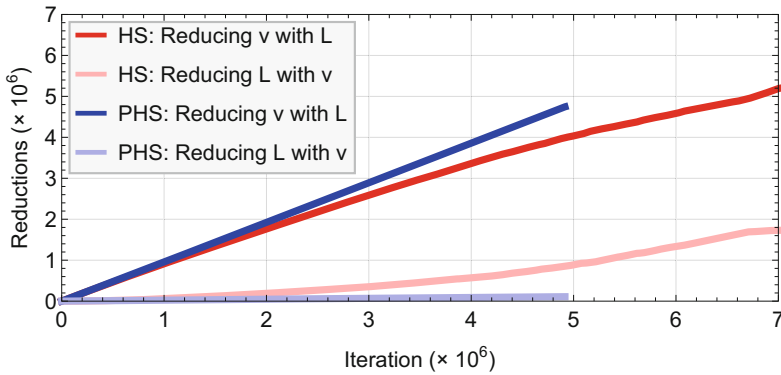


(c) **Lattice rank.** For the ProHashSieve, the rank of the sublattice (roughly) follows a square-root law, with more time spent on higher ranks.

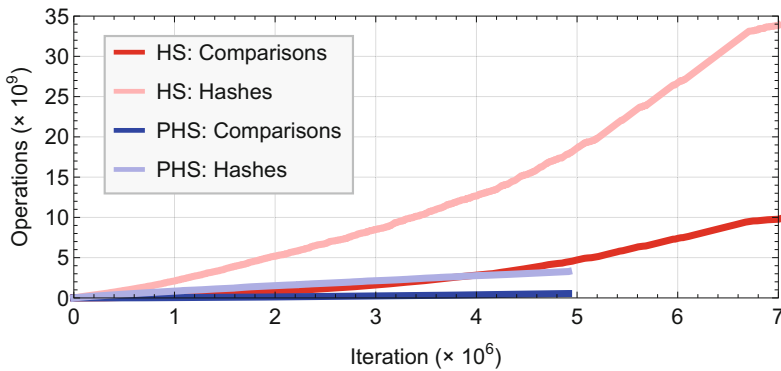
Fig. 2. Sieving profiles for the HashSieve (red) and ProHashSieve (blue), when given as input a BKZ-30 reduced lattice basis of a 70-dimensional lattice. (Color figure online)



(a) **Norm of the shortest vector.** The dashed black curve shows the theoretical prediction for the ProHashSieve based on the GSA and rank.



(b) **Reductions ($\times 10^6$).** Progressive sieving rarely sees reductions of the list with new vectors, making the GaussSieve similar to the ListSieve.



(c) **Operations ($\times 10^9$).** Both algorithms roughly have the same ratio of hashing operations (finding near neighbors) and vector comparisons.

Fig. 3. Sieving profiles for the HashSieve (red) and ProHashSieve (blue), when given as input a BKZ-30 reduced lattice basis of a 70-dimensional lattice. (Color figure online)

Finally, to illustrate the improvements for finding approximate shortest vectors, Table 2 lists the time complexities for 1.1-SVP on 70-dimensional lattices from the SVP challenge [SVP18]: the measured complexities are based on terminating the algorithm as soon as a vector $\mathbf{v} \in \mathcal{L}$ of norm $\|\mathbf{v}\| \leq 1.1 \cdot \lambda_1(\mathcal{L})$ has been found. As the results show (and as could already be seen in Fig. 3a), standard sieving barely benefits from this relaxed termination requirement, while progressive sieving improves considerably both when an approximate solution suffices, and when the input bases is more reduced, leading to extreme speedup factors as high as $5000\times$. This highlights a weakness of existing lattice sieving methods.

Table 2. Running times (in seconds) for solving approximate SVP with approximation factor $\gamma = 1.1$ on 70-dimensional lattice bases from the SVP challenge [SVP18]. When the bases are sufficiently well-reduced, low-rank sublattices already contain very short lattice vectors, leading to substantially faster convergence for progressive sieving.

Approximate SVP ($\gamma = 1.1$)	← GaussSieve →			← HashSieve →		
	LLL	BKZ-10	BKZ-30	LLL	BKZ-10	BKZ-30
Standard sieving	18500	17200	15600	3180	2960	2700
Progressive sieving	120	40	3	65	20	2
Speedup factor	150 ×	400 ×	5000 ×	50 ×	150 ×	1000 ×

5 Discussion

In this section we will briefly discuss different aspects of lattice sieving, and how various design choices affect the performance of (progressive) sieving in practice.

5.1 Approximate SVP

As shown in Table 2, progressive sieving benefits greatly from being given a relaxed termination condition, in the sense that an approximate solution to SVP suffices. To quantify where this improvement comes from, recall that by the Geometric Series Assumption (GSA), BKZ gives us a reduced lattice basis $\{\mathbf{b}_1, \dots, \mathbf{b}_d\}$ where the Gram-Schmidt orthogonalization vectors satisfy $\|\mathbf{b}_i^*\| \approx \delta^{d-2i+1} \cdot \text{Vol}(\mathcal{L})^{1/d}$ for all $i = 1, \dots, d$. As a result, the first k basis vectors together form a rank- k sublattice $\mathcal{L}_k = \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_k) \subset \mathcal{L}$ of volume:

$$\text{Vol}(\mathcal{L}_k) = \prod_{i=1}^k \|\mathbf{b}_i^*\| = \text{Vol}(\mathcal{L})^{k/d} \cdot \delta^{\sum_{i=1}^k (d-2i+1)} = \text{Vol}(\mathcal{L})^{k/d} \cdot \delta^{k(d-k)} \quad (1)$$

By the Gaussian heuristic we have $\lambda_1(\mathcal{L}) \approx \text{Vol}(\mathcal{L})^{1/d} / \sqrt{2e\pi d}$, and therefore the shortest vector in this rank- k sublattice \mathcal{L}_k has relative norm:

$$\lambda_1(\mathcal{L}_k) \approx \frac{\text{Vol}(\mathcal{L}_k)^{1/k}}{\sqrt{2e\pi k}} = \frac{\delta^{d-k} \text{Vol}(\mathcal{L})^{1/d}}{\sqrt{2e\pi k}} \approx \delta^{d-k} \sqrt{\frac{d}{k}} \cdot \lambda_1(\mathcal{L}) \quad (2)$$

To obtain a solution to γ -SVP, i.e. approximate SVP with approximation factor γ , we need $\delta^{d-k} \sqrt{d/k} \approx \gamma$. Depending on the basis quality δ , dimension d , and approximation factor γ , this tells us how many dimensions we can essentially “skip” at the end when looking for approximate solutions: already at rank k , we will then expect to find a vector of sufficiently short length.

One could argue that the picture sketched by Table 2 is painted too rosily in favor of progressive sieving, since classical sieving approaches could also find approximate solutions faster by (a) computing a similar heuristic on what rank k is required to find an approximate solution in a suitable sublattice, and then (b) simply running sieving on such a sublattice until an approximate solution has been found. Even in that case however, progressive sieving has the benefits of not requiring an a priori choice of k (it automatically finds the right rank k required to find a sufficiently short vector), and having a faster convergence for exact SVP in such sublattices as well (as sketched in Table 1).

5.2 Effects of Basis Reduction

The experiments in Sect. 4 further suggest that progressive sieving benefits more from being given a more reduced basis than classical sieving approaches. To explain this, we observe that there are two sides to having better bases, and previous sieving approaches only benefited from one of these advantages.

Standard sieving. When given a better basis, classical sieving methods will be able to sample shorter lattice vectors in Line 4 more easily. Being able to sample shorter vectors means that vectors will need to be reduced less frequently before “stabilizing” in the list, thus reducing the overall cost of the algorithm as for each vector, we might save some searches and reductions.

Progressive sieving. Besides the benefit stated above, which also holds for progressive sieving, there is a second advantage to being given a nice lattice basis. This is closely related to the heuristic analysis for approximate SVP above, as having a reduced basis means that δ is smaller and therefore low-rank sublattices will already contain shorter lattice vectors. And being able to find very short lattice vectors early in low ranks, which will then be contained in the list L when moving to higher-rank sublattices, means that reductions will proceed faster in those higher ranks as well.

Formally, let $\mathcal{S}^{d-1} = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = 1\}$ denote the unit sphere, and let $\mathbf{v} \sim \mathcal{S}^{d-1}$ be sampled uniformly at random from this sphere. Then, using spherical cap arguments similar to e.g. [BDGL16], one can show that the probability that \mathbf{v} can be reduced with a random vector $\mathbf{w} \in \mathcal{S}^{d-1}$ (i.e. $\|\mathbf{v} \pm \mathbf{w}\| \leq \|\mathbf{v}\|$) is $(3/4)^{d+o(d)} \approx 2^{-0.21d+o(d)}$. However, when the norms of list vectors \mathbf{w} differ from 1, so that the norms of \mathbf{v} and \mathbf{w} are different, this probability changes. Intuitively, for small $\|\mathbf{w}\| \rightarrow 0$ the probability of reducing \mathbf{v} with \mathbf{w} approaches 1, as either $\mathbf{v} + \mathbf{w}$ or $\mathbf{v} - \mathbf{w}$ is likely to be shorter than \mathbf{v} . For large $\|\mathbf{w}\| \rightarrow \sqrt{2}\|\mathbf{v}\|$ the probability of being able to reduce \mathbf{v} with \mathbf{w} approaches 0, and for $\|\mathbf{w}\| \geq 2\|\mathbf{v}\|$

it is even impossible for such a reduction to take place. For arbitrary $\alpha \in (0, \sqrt{2})$ and random $\mathbf{w} \in \alpha \mathcal{S}^{d-1}$, a geometric exercise shows that this probability scales as $(1 - \alpha^2/2)^{d/2+o(d)}$, showing that if list vectors $\mathbf{w} \in L$ are a factor α shorter than the sampled vectors \mathbf{v} , the likelihood that \mathbf{v} can be reduced with the list is exponentially larger (in α) than when vectors in the list have equal norm as \mathbf{v} .

Summarizing, if the input basis to the sieve is more reduced, then in lower ranks the algorithm will already find shorter lattice vectors to add to the list L , and having shorter vectors in the list means that new vectors can be reduced faster and more easily. Since in classical sieving approaches it takes much longer to find (approximate) short lattice vectors, previous approaches to sieving do not benefit from this accelerated reductions when the list vectors are short. We believe this mostly explains the bigger improvements to progressive sieving when using more reduced bases.

Pruning. Note that when the lattice basis is very reduced, the last coefficients of the shortest lattice vector in terms of these basis vectors are commonly small or zero. If the search space in the last few coefficients is small, one could simply guess or enumerate these potential coefficients, while doing sieving on the first coefficients. This suggests an approach where we only do sieving up to a certain rank, and for the last few basis vectors we use a different procedure of either guessing the coefficients and randomizing/restarting when we fail (similar to pruned enumeration with randomized bases), or using e.g. enumeration or Babai rounding to deal with these latter dimensions. A somewhat similar idea of dealing with these last dimensions faster was analyzed in [Duc18].

5.3 Effects of the Sampler

By the sampler, we are referring to the procedure in Line 4 of Algorithms 1 and 2 of sampling new lattice vectors, in case the stack is empty: a new vector is sampled from a certain distribution over the lattice, using some efficient sampling algorithm. This sampling is often done through a randomized enumeration procedure [Kle00], and the exact specification of this procedure determines how short the sampled lattice vectors will be, how often similar vectors (in particular, the vector $\mathbf{0}$) are sampled, and how long the procedure takes to produce a new sample.

Standard sieving. Choosing the best parameters for the sampler, to get the best performance for sieving, is a rather cumbersome procedure: there are several parameters to tune in the sampler alone, and the best choice varies per dimension and per lattice sieve method (GaussSieve, HashSieve, etc.). Experiments in the past have shown that for previous sieving approaches, this choice of parameters may also greatly influence the practical performance of the algorithm [IKMT14, FBB+14, MLB15, MB16], which makes this aspect of sieving hard to deal with properly – choosing parameters optimally is both difficult and important.

Progressive sieving. Although choosing parameters accurately still influences the performance of progressive sieving, sampling longer vectors causes less of a slowdown due to the list having many short vectors early on (as discussed before, in the context of the input basis quality). Long sampled vectors are reduced in length faster and more easily, so that sampling longer vectors is less of an issue. Therefore fine-tuning the sampler will likely not lead to as big improvements for progressive sieving as for standard sieving. However, this also means that the numbers in Tables 1, 2 and Fig. 1 may not be entirely accurate when using optimized samplers – our numbers are based on a simple, straightforward proof-of-concept implementation of Klein’s sampler, rather than having optimized all aspects of the algorithm (and in particular this sampling routine).²

5.4 Effects of List Updates (GaussSieve vs. ListSieve)

As mentioned, the main difference between Micciancio–Voulgaris’ GaussSieve and ListSieve is that in the ListSieve, vectors that are added to the list are never modified again. By “list updates”, we are referring to either making updates to the list vectors as in the GaussSieve, or never updating list vectors as in the ListSieve algorithm.

Standard sieving. In classical sieving approaches, as can for instance be seen in Fig. 3b, lattice vectors in the sieved list are quite often reduced and moved back to the stack, after which they are processed again and pushed back to the list. In the ListSieve-variant of such algorithms, where list vectors are never touched again, one would miss all these reductions, leading to a significantly worse performance overall [MODB14]. Although the ListSieve is conceptually slightly simpler, the performance loss would not be worth it to ever consider using the ListSieve instead of the GaussSieve in high-performance environments.

Progressive sieving. For progressive sieving, reductions of list vectors almost never happen, as can also be seen in Fig. 3b. Recall that the experiments described in Figs. 2 and 3 are based on the GaussSieve-based HashSieve, where such list updates are allowed if they produce shorter lattice vectors. For progressive sieving, the execution times of the GaussSieve and ListSieve variants are much closer, which suggests using ListSieve-like sieving may be practical as well. Note that the ListSieve does not save any inner product computations between sampled and list vectors, since we already need to compute $\mathbf{v} \cdot \mathbf{w}$ to see if \mathbf{v} can be reduced with \mathbf{w} in Line 5 (before reducing \mathbf{w} with \mathbf{v} in Line 6), and so the performance is still worse when doing the ListSieve instead of the GaussSieve: skipping potential reductions which are “free” is almost never a wise choice.

² Concurrent work [Duc18] suggests that ideas similar to progressive sieving only lead to a factor $5\times$ speed-up, i.e. roughly a factor $4\times$ less than described here. We conjecture that this difference is mainly caused by Ducas using a more optimized implementation of the baseline approach, and in particular using a better sampler.

A potential application of ListSieve-style sieving, knowing that its performance is closer to the GaussSieve when using progressive sieving, is in parallel implementations. For shared memory systems, current approaches use (probable) lock-free lists [MTB14,MLB15,MLB17], and these lock-free lists may no longer be needed when we know that once a vector is added to the list, it is never updated again and it becomes read-only memory. For distributed-memory implementations [IKMT14,BNvdP16,YKYC17], using a ListSieve as a baseline may also be beneficial in terms of performance and simplicity – synchronization between nodes only has to be done on local sampled vectors, and not on list vectors which are modified by different nodes.

6 Open Problems

As we have seen, progressive sieving has a better (experimental) performance than current sieving approaches in many ways, making sieving slightly smoother, more predictable, less dependent on parameter choices for the sampler, and more dependent on the quality of the input basis. Below we state some open problems for future work, related to the ideas presented in this paper.

6.1 BKZ and Sieving on Sliding Windows

A long-standing open problem is to efficiently implement sieving as the SVP subroutine for BKZ, which now commonly uses enumeration as its SVP subroutine instead. Only recently has sieving become rather competitive with enumeration, and projects in this direction are likely ongoing.

When running BKZ on d -dimensional bases, there is often a sliding window of k indices $i+1, \dots, i+k$, for which SVP in k dimensions needs to be solved to form an HKZ-reduced basis in this block. Then, when this block has been properly reduced, the index i is increased by 1, and the same procedure is applied to the window $i+2, \dots, i+k+1$. Since there is a large overlap between different windows, a natural question is whether dealing with this new block can be done more efficiently than starting from scratch.

This is somewhat similar to progressive sieving, where as an abstraction we start with k basis vectors $\mathbf{b}_1, \dots, \mathbf{b}_k$, and after reducing this basis (running sieving on this block), we switch to a basis $\pi_1(\mathbf{b}_2), \dots, \pi_1(\mathbf{b}_{k+1})$, with $\pi_1(\mathbf{x})$ being the projection of \mathbf{x} orthogonal to \mathbf{b}_1 , and \mathbf{b}_{k+1} being linearly independent of the previous basis vectors. So (besides the projections) not only do we add a new basis vector \mathbf{b}_{k+1} to the system as in progressive sieving, we also remove one vector \mathbf{b}_1 , making all vectors currently in our list with a non-zero coefficient of \mathbf{b}_1 “unusable” in the next iteration.

One potential way of dealing with shifting windows in sieving is to discard all vectors \mathbf{w} with non-zero coefficient of \mathbf{b}_1 when moving from one window to the next – the contribution of \mathbf{b}_1 may be crucial for this vector to be short, and removing this contribution may result in a long lattice vector. By the GSA, the number of vectors of norm less than $\frac{4}{3} \cdot \lambda_1(\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_k))$ in $\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_k)$

is approximately $[\frac{4}{3} \cdot \lambda_1(\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_k)) / \lambda_1(\mathcal{L}(\mathbf{b}_2, \dots, \mathbf{b}_k))]^d$, which is roughly a fraction $[\lambda_1(\mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_k)) / \lambda_1(\mathcal{L}(\mathbf{b}_2, \dots, \mathbf{b}_k))]^d$ of all vectors in the sieved list $L \subset \mathcal{L}(\mathbf{b}_1, \dots, \mathbf{b}_k)$. Although this fraction may be small depending on the shape of the lattice and the block size k , a non-negligible fraction of vectors can thus be reused in the next iteration without difficulties.

Besides this straightforward approach, one could also imagine vectors with non-zero coefficients of \mathbf{b}_1 being reused in the next iteration by just removing this contribution of \mathbf{b}_1 , and hoping that the resulting vector is still short. This might make the vectors slightly longer on average, but makes sure that all vectors can be reused, potentially saving even more work on the sieve in the next window.

6.2 Enumeration with Sieving

Another long-standing open problem in lattice algorithms is to consider approaches based on enumeration and sieving, and combining the “best of both worlds” to construct even better SVP algorithms. As suggested in [Laa16], one potential such combination would consist of running sieving on a subset of the basis (say $\mathbf{b}_1, \dots, \mathbf{b}_k$), and then using the sieved list as an approximate Voronoi cell for faster enumeration. This enumeration procedure would then consist of considering combinations of the vectors $\mathbf{b}_{k+1}, \dots, \mathbf{b}_d$ as in state-of-the-art enumeration algorithms, and then using the sieved list $L \subset \mathcal{L}_k$ to see whether those enumerated vectors are close to a vector in the sublattice \mathcal{L}_k . If it is close to such a vector, the difference with that vector is likely a short vector in the full lattice. In this case sieving would be used as a batch-CVP/CVPP oracle.

While this idea also works with classical sieving methods, it becomes even more natural with progressive sieving, which already considers increasingly large sublattices to make progress. Modifying progressive sieving to the above application would simply mean changing the upper bound in Line 16 from the full rank d to some bound d_0 . Similar ideas of combining sieving on a sublattice with enumerating the “last few dimensions” have been studied in [Duc18], but more work is needed to understand the full potential of such a hybrid approach.

Acknowledgments. The authors thank Léo Ducas for discussions and comments on this topic, and for sharing an early draft of [Duc18]. The first author is supported by the ERC consolidator grant 617951. The second author was partially supported by Fundação para a Ciência e a Tecnologia (FCT) and Instituto de Telecomunicações under grant UID/EEA/50008/2013.

References

- [ADPS16] Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: USENIX Security Symposium, pp. 327–343 (2016)
- [AKS01] Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: STOC, pp. 601–610 (2001)

- [AN17] Aono, Y., Nguyen, P.Q.: Random sampling revisited: lattice enumeration with discrete pruning. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 65–102. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_3
- [BDGL16] Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: SODA, pp. 10–24 (2016)
- [BDK+18] 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. In: EuroS&P (2018)
- [BGJ14] Becker, A., Gama, N., Joux, A.: A sieve algorithm based on overlattices. In: ANTS, pp. 49–70 (2014)
- [BGJ15] Becker, A., Gama, N., Joux, A.: Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522, pp. 1–14 (2015)
- [BL16] Becker, A., Laarhoven, T.: Efficient (Ideal) lattice sieving using cross-polytope LSH. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2016. LNCS, vol. 9646, pp. 3–23. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31517-1_1
- [BLS16] Bai, S., Laarhoven, T., Stehlé, D.: Tuple lattice sieving. In: ANTS, pp. 146–162 (2016)
- [BNvdP16] Bos, J.W., Naehrig, M., van de Pol, J.: Sieving for shortest vectors in ideal lattices: a practical perspective. *Int. J. Appl. Cryptogr.* **3**, 1–23 (2016)
- [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
- [DLL+18] Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - dilithium: digital signatures from module lattices. In: CHES (2018)
- [Duc18] Ducas, L.: Shortest vector from lattice sieving: a few dimensions for free. In: EUROCRYPT (2018)
- [FBB+14] Fitzpatrick, R., Bischof, C., Buchmann, J., Dagdelen, Ö., Göpfert, F., Mariano, A., Yang, B.-Y.: Tuning GaussSieve for speed. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014. LNCS, vol. 8895, pp. 288–305. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16295-9_16
- [FP85] Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice. *Math. Comput.* **44**(170), 463–471 (1985)
- [fpLLL18] The FPLLL Development Team. FPLLL, a lattice reduction library (2016). <https://github.com/fplll/fplll>
- [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
- [HK17] Herold, G., Kirshanova, E.: Improved algorithms for the approximate k -list problem in euclidean norm. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10174, pp. 16–40. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54365-8_2
- [HKL18] Herold, G., Kirshanova, E., Laarhoven, T.: Speed-ups and time-memory trade-offs for tuple lattice sieving. In: PKC (2018)

- [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
- [IKMT14] Ishiguro, T., Kiyomoto, S., Miyake, Y., Takagi, T.: Parallel gauss sieve algorithm: solving the SVP challenge over a 128-dimensional ideal lattice. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 411–428. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_24
- [Kan83] Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: STOC, pp. 193–206 (1983)
- [Kle00] Klein, P.: Finding the closest lattice vector when it’s unusually close. In: SODA, pp. 937–941 (2000)
- [Laa15] Laarhoven, T.: Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 3–22. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_1
- [Laa16] Laarhoven, T.: Finding closest lattice vectors using approximate Voronoi cells. Cryptology ePrint Archive, Report 2016/888 (2016)
- [LdW15] Laarhoven, T., de Weger, B.: Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 101–118. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_6
- [MB16] Mariano, A., Bischof, C.: Enhancing the scalability and memory usage of HashSieve on multi-core CPUs. In: PDP, pp. 545–552 (2016)
- [MLB15] Mariano, A., Laarhoven, T., Bischof, C.: Parallel (probable) lock-free HashSieve: a practical sieving algorithm for the SVP. In: ICPP, pp. 590–599 (2015)
- [MLB17] Mariano, A., Laarhoven, T., Bischof, C.: A parallel variant of LDSieve for the SVP on lattices. In: PDP (2017)
- [MLC+17] Mariano, A., Laarhoven, T., Correia, F., Rodrigues, M., Falcao, G.: A practical view of the state-of-the-art of lattice-based cryptanalysis. IEEE Access **5**, 24184–24202 (2017)
- [MODB14] Mariano, A., Dagdelen, Ö., Bischof, C.: A comprehensive empirical comparison of parallel ListSieve and GaussSieve. In: Lopes, L., et al. (eds.) Euro-Par 2014. LNCS, vol. 8805, pp. 48–59. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14325-5_5
- [MS11] Milde, B., Schneider, M.: A parallel implementation of GaussSieve for the shortest vector problem in lattices. In: Malyshev, V. (ed.) PaCT 2011. LNCS, vol. 6873, pp. 452–458. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23178-0_40
- [MTB14] Mariano, A., Timnat, S., Bischof, C.: Lock-free Gauss-Sieve for linear speedups in parallel high performance SVP calculation. In: SBAC-PAD, pp. 278–285 (2014)
- [MV10a] Micciancio, D., Voulgaris, P.: A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In: STOC, pp. 351–358 (2010)
- [MV10b] Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: SODA, pp. 1468–1480 (2010)
- [NV08] Nguyễn, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. J. Math. Cryptol. **2**(2), 181–207 (2008)

- [Sch87] Schnorr, C.-P.: A hierarchy of polynomial time lattice basis reduction algorithms. *Theoret. Comput. Sci.* **53**(2–3), 201–224 (1987)
- [Sch11] Schneider, M.: Analysis of Gauss-Sieve for solving the shortest vector problem in lattices. In: Katooh, N., Kumar, A. (eds.) *WALCOM 2011*. LNCS, vol. 6552, pp. 89–97. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19094-0_11
- [Sch13] Schneider, M.: Sieving for shortest vectors in ideal lattices. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) *AFRICACRYPT 2013*. LNCS, vol. 7918, pp. 375–391. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38553-7_22
- [SE94] Schnorr, C.-P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Math. Program.* **66**(2–3), 181–199 (1994)
- [SFS09] Sommer, N., Feder, M., Shalvi, O.: Finding the closest lattice point by iterative slicing. *SIAM J. Discret. Math.* **23**(2), 715–731 (2009)
- [SVP18] SVP Challenge (2018). <https://latticechallenge.org/svp-challenge/>
- [WLTB11] Wang, X., Liu, M., Tian, C., Bi, J.: Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In: *ASIACCS*, pp. 1–9 (2011)
- [YKYC17] Yang, S.-Y., Kuo, P.-C., Yang, B.-Y., Cheng, C.-M.: Gauss sieve algorithm on GPUs. In: Handschuh, H. (ed.) *CT-RSA 2017*. LNCS, vol. 10159, pp. 39–57. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_3
- [ZPH13] Zhang, F., Pan, Y., Hu, G.: A three-level sieve algorithm for the shortest vector problem. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) *SAC 2013*. LNCS, vol. 8282, pp. 29–47. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_2



A Nonstandard Variant of Learning with Rounding with Polynomial Modulus and Unbounded Samples

Hart Montgomery^(✉)

Fujitsu Laboratories of America, Sunnyvale, USA
hmontgomery@us.fujitsu.com

Abstract. The *Learning with Rounding Problem* (LWR) has become a popular cryptographic assumption to study recently due to its determinism and resistance to known quantum attacks. Unfortunately, LWR is only known to be provably hard for instances of the problem where the LWR modulus q is at least as large as some polynomial function of the number of samples given to an adversary, meaning LWR is provably hard only when (1) an adversary can only see a fixed, predetermined amount of samples or (2) the modulus q is superpolynomial in the security parameter, meaning that the hardness reduction is from superpolynomial approximation factors on worst-case lattices.

In this work, we show that there exists a (still fully deterministic) variant of the LWR problem that allows for both unbounded queries and a polynomial modulus q , breaking an important theoretical barrier. To our knowledge, our new assumption, which we call the *Nearby Learning with Lattice Rounding Problem* (NLWLR), is the first fully deterministic version of the learning with errors (LWE) problem that allows for both unbounded queries and a polynomial modulus. We note that our assumption is not practical for any kind of use and is mainly intended as a theoretical proof of concept to show that provably hard deterministic forms of LWE can exist with a modulus that does not grow polynomially with the number of samples.

Keywords: Lattices · LWE · Learning with rounding

1 Introduction

In recent years, the need for quantum-secure cryptographic protocols has seemingly dramatically increased. Many people and organizations, including apparently the NSA, believe that powerful quantum computers will be coming soon. It therefore seems imperative that the cryptographic community develop new, efficient quantum-secure protocols for all of the common use cases of cryptography today.

The full version of this paper is available on the IACR cryptology eprint archive.

Most of these protocols are based upon the *Learning with Errors Assumption* (LWE), which was invented by Regev in [Reg05], or its ring variant. Informally, given a fixed random vector \mathbf{s} , the LWE assumption states that it is difficult to distinguish samples of the form $(\mathbf{a}_i \mathbf{a}_i^\top \mathbf{s} + \delta_i)$ from random when the \mathbf{a}_i s are uniformly sampled random vectors and the δ_i s are fresh samples from a low-norm noise distribution. While it is well-known that the LWE assumption has played a huge role in the development of theoretical cryptography, including things like fully homomorphic encryption [BV11], recently LWE has been gaining traction as an assumption for practical cryptography due to the need for quantum security, including in [BCD+16, Pei14]. In fact, it seems like most of the submissions to the NIST post-quantum key exchange competition will be based on LWE (or its ring variants).

But with all of the attention given to the LWE assumption, we think it is worth considering a related, simpler, and more efficient assumption: the learning with rounding (LWR) assumption. The LWR assumption was invented in [BPR12] as a way to build the first nontrivial, parallelizable (and hence low depth) PRFs¹ from lattice assumptions. Informally, given a fixed random vector \mathbf{s} , the LWR assumption states that it is difficult to distinguish samples of the form $(\mathbf{a}_i \lceil \mathbf{a}_i^\top \mathbf{s} \rceil_p)$ from random when the \mathbf{a}_i s are random vectors sampled uniformly at random. Note that the operation $\lceil \cdot \rceil_p$ can be thought of as the general rounding operation ‘round to the nearest multiple of $\frac{q}{p}$ and then divide by $\frac{q}{p}$ ’. In the vast majority of cases where the LWE assumption is used, the LWR assumption can be used in its place.

LWR has several advantages over LWE. The most obvious one is that, for fixed dimension and vector distributions (i.e. key and sample distributions), LWR is faster to compute. This is due to the fact that we don’t need to sample from the noise distribution when generating an LWR sample, and this noise sampling is typically a major pain point for practical LWE implementations. But LWR is also more resilient than LWE in many ways as well: since it is deterministic, it makes it so an adversary that can somehow trick an oracle into repeating samples (outputting a sample with the same value of \mathbf{a}_i but with a different noise sample) gains nothing. An often-overlooked fact about LWR is that it would likely be much more resistant to side channel attacks than LWE, since noise sampling can be difficult and take a lot of effort to inure against these sorts of attack [RRVV14]. This fact implies that LWR would be a very good post-quantum candidate for certain hardware implementations.

There have been many recent schemes that utilized the power of LWR. A natural use case was lattice-based PRFs, including [BLMR13, BP14]. Papers with a practical focus on things like key exchange have also been build using rounding, including [CKLS16, DFH+16]. More powerful PRFs that provide increased functionality and also might provide illustrating examples that even touch at

¹ It was previously known how to build completely sequential (and thus high depth) PRFs from PRGs using generic constructions like [GGM84]. It is possible to build a very simple lattice-based PRF using the [GGM84] construction by treating LWE as a PRG.

things like indistinguishability obfuscation include papers like [BFP+15, CC17, BKM17].

Given all of the benefits of LWR mentioned so far, it seems silly that anyone would ever want to use the LWE assumption instead. However, there is an excellent reason why people use LWE instead of LWR: security. As of now, LWE does not reduce tightly to LWR, and LWR does not have a direct reduction from worst-case lattice problems like LWE does. To reach the same provable security levels as LWE schemes, it is often the case that the parameters of LWR schemes have to be made substantially larger than their LWE counterparts, negating many of the advantages of LWR.

Fundamentally, we think that there does not seem to be any obvious reason why LWR should not be as hard or close to as hard as LWE (although without proofs we can never be sure)². The goal of this work is to continue to try to close the gap between the two problems in terms of hardness. In this work, we do not fully close this gap, or even come up with a primitive that is practically useful—we cannot think of any practical use cases for what we build in this paper. But we do make progress on what we consider an extremely important practical problem in lattice cryptography.

In fact, we think that if the hardness gap between LWR and LWE was closed, then there would be little use for LWE in practice. For instance, if LWR was provably secure for parameters where the ‘rounding loss’ $\frac{q}{p}$ was not much larger than the noise magnitude for LWE with similar levels of security, many of the very practical NIST post-quantum crypto proposals like Kyber [BDK+17] or Frodo [BCD+16] would probably be redesigned to use LWR. We think that research on the hardness of LWR (or the lack thereof) could have widespread practical application in the long run.

1.1 LWR: Security Background

In order to put our work in context and explain the motivation, it is useful to go over the previous work on LWR and LWR security.

Original LWR Work [BPR12]: the original LWR paper [BPR12] proved the hardness of LWR where the modulus q was superpolynomially larger than the B -bounded noise distribution used in the LWE oracle in the proof. The reduction was straightforward: take an LWE oracle, and round the LWE output. If the rounded LWE samples $(\mathbf{a}_i, \lceil \mathbf{a}_i^\top \mathbf{s} + \delta_i \rceil_p)$ were always equal to the LWR samples $(\mathbf{a}_i, \lceil \mathbf{a}_i^\top \mathbf{s} \rceil_p)$, then the LWR problem was at least as hard as the underlying LWE problem. While this explanation simplifies the reduction, this is the fundamental idea of how it works in this paper.

However, this condition obviously does not hold if the error δ_i causes the rounded LWE output $\lceil \mathbf{a}_i^\top \mathbf{s} + \delta_i \rceil_p$ to differ in \mathbb{Z}_p from the rounded LWR output $\lceil \mathbf{a}_i^\top \mathbf{s} \rceil_p$. Since we have no way of authoritatively telling whether this bad event

² Caution! This is just an opinion.

happened or not, the authors of [BPR12] required that the modulus q be super-polynomial relative to the B -bounded noise distribution used in the LWE oracle. This ensured that the probability that any $\mathbf{a}_i^\top \mathbf{s}$ was within B of a ‘boundary’ where the rounding value changed was negligible for any polynomial amount of LWR samples and allowed the proof to work.

Subsequent Work. After the original paper, several further works cleverly improved the state of LWR [AKPW13, BGM+16, BLL+15, AA16]. In general, the authors of these papers showed that, by cleverly sampling and leaking certain secret information and using smart statistical analyses, it was possible to avoid instances where rounded LWE and LWR samples differed in the reduction. These works substantially improved the parameters for LWR in general, but unfortunately still required an a priori bounded number of samples in the case of a polynomial modulus. Let c be a constant³, $\gamma \geq 1$, let B be the B -bound for the noise distribution of the LWE instance that we reduce to our LWR instance, let p be the rounding parameter (the group \mathbb{Z}_p that we are rounding to), and let κ be a security parameter. The table below (with format borrowed from [AA16]) summarizes the state of the art.

Work	Unbounded samples (w)	Modulus (q)	Advantage change ($\varepsilon \rightarrow \varepsilon'$)
[BPR12]	Yes	$Bp\kappa^{\omega(1)}$	$\varepsilon - \text{negl}(\kappa)$
[AKPW13]	No	$\gamma Bwp\kappa$	$\varepsilon/(2dw)$
[BGM+16]	No	Bwp	$(\varepsilon/qw)^2$
[BLL+15]	No	Bwp	$(\varepsilon/qw)^2$
[AA16] (1)	No	$Bwp\kappa$	$\varepsilon(wB)^{-c}$
[AA16] (2)	No	$Bwp\kappa$	$\varepsilon(w)^{-c}$

Work	Dimension change ($d \rightarrow d'$)	Straightforward rounding	Uniform samples
[BPR12]	d	Yes	Yes
[AKPW13]	$d \log(\gamma)/\log q$	Yes	Yes
[BGM+16]	$d/\log_\rho q$	Yes	Yes
[BLL+15]	d	Yes	Yes
[AA16] (1)	d	Yes	Yes
[AA16] (2)	$d - c$	Yes	Yes

³ For typical choices of parameters.

A First Attempt. When attempting to reduce LWE to LWR, it is natural to ask why we cannot just throw out or ignore the samples that are close to these rounding ‘boundaries’. We could use probabilistic rejection sampling to make our LWR sample distribution mimic the output of an LWE oracle. This works in essentially the following way (we are leaving out some details here, and what we say isn’t exactly correct, but the intuition remains true): recall that a regular LWR oracle would simply just output samples of the form $(\mathbf{a}_i, \lceil \mathbf{a}_i^\top \mathbf{s} \rceil_p)$. Now suppose we have a new oracle that outputs samples in a somewhat similar manner, but with a catch: for every sample, it samples some δ_i from the LWE noise distribution ψ . If $\mathbf{a}_i^\top \mathbf{s} + \delta_i$ is within a distance of B from the rounding boundary, then we reject the sample and start over. Otherwise, we go on ahead and output $(\mathbf{a}_i, \lceil \mathbf{a}_i^\top \mathbf{s} \rceil_p)$ as usual.

It shouldn’t take too much effort to see that the behavior of this new LWR oracle is exactly the same as the following oracle: Take samples from an LWE oracle, and round the second term, getting a term of the form $(\mathbf{a}_i, \lceil \mathbf{a}_i^\top \mathbf{s} + \delta_i \rceil_p)$, but reject the samples if $\mathbf{a}_i^\top \mathbf{s} + \delta_i$ is within B of a rounding boundary. This oracle is directly simulatable given an LWE oracle, so we can prove hardness directly from LWE using a reduction like this.

Unfortunately, this approach still has one glaring hole: this type of rounding with probabilistic rejection is not fully deterministic. It is true that, for a fixed instance of the problem, every tuple whose first term is \mathbf{a}_i will always have second term $\lceil \mathbf{a}_i^\top \mathbf{s} \rceil_p$. However, which $\mathbf{a}_i \mathbf{s}$ are in fact included in samples is highly nonuniform—and even probabilistic—and even though it is very simple to show that the accumulated distribution of $\mathbf{a}_i \mathbf{s}$ is computationally hard to distinguish from fully random for one instance of the problem, this still presents significant problems for many applications that require determinism. For instance, if we attempt to build a PRF using this technique, different instantiations of the PRF may have different outputs, since, depending on the noise sampled, some instantiations of the PRF will reject certain $\mathbf{a}_i \mathbf{s}$ while others will accept them. This could be disastrous for security if an adversary has access to multiple copies of the PRF, since we can guess how close a particular value of $\mathbf{a}_i^\top \mathbf{s}$ is to the boundary based on the number of outputs versus rejections for a particular sample.

It seems difficult to improve this basic attempt if we wanted to keep the number of queries to be an unbounded polynomial, since we cannot know any extra information about $\mathbf{a}_i^\top \mathbf{s}$ in the simulation, and we could not leak any non-negligible amount of information per query to help us without risking a complete reveal of the secret \mathbf{s} .

Lattice Rounding. Our first core idea is the following: what if, instead of rounding each sample to the nearest multiple of some integer, we group samples together and deterministically round them to a nearby lattice point (not necessarily the nearest, obviously)? We might be able to leak more information that can help us eliminate rounding mistakes this way, as it could potentially be hard for an

adversary to determine whether or not a point is close to a boundary created by whatever lattice rounding algorithm we are using (think of Babai’s nearest plane algorithm or regular Babai rounding)⁴.

However, we have to be careful when doing this. A simulator or adversary cannot have access to both the values of uniformly random $\mathbf{a}_i^\top \mathbf{s}$ terms and the lattice points gained as a result of rounding, or otherwise something similar to the famous ‘learning a hidden parallelepiped’ attack in [NR06] seems to occur and the (short) basis of the lattice can be learned. Once a simulator (or an adversary for that matter) has access to the basis we are using to round, then we gain very little from rounding to a lattice instead of to the nearest multiple of some integer. Perhaps some clever trick could thwart this class of attacks, but we could not come up with such an idea.

Restricting the Samples. The next main idea is something that arises naturally from our first attempt: suppose we continue to utilize ‘lattice rounding’, but what if we restrict the possible values of the samples \mathbf{A}_i such that the values $\mathbf{A}_i \mathbf{s}$ are in some kind of ball around points in the lattice that we are rounding to rather than let them be uniform? Note that our samples consist of matrices now instead of vectors. We can use the LWE assumption to show that such a distribution looks random to an adversary, and we can simulate such samples without a trapdoor for the lattice we are rounding to as well: pick a random point in the lattice to which we will ‘round’ the final value $\mathbf{A}_i \mathbf{s}$ (given a ‘bad’ basis, of course), add noise to get the ‘actual’ value of $\mathbf{A}_i \mathbf{s}$, and then output the value of \mathbf{A}_i such that $\mathbf{A}_i \mathbf{s}$ equals the lattice point we picked plus the noise term.

This has the unfortunate consequence that we cannot use a uniform distribution of $\mathbf{A}_i \mathbf{s}$, but allows us to potentially prove security because we do not need to know a short basis of the lattice we are rounding to in order to simulate queries. However, we are still unable to issue a challenge query in this regime—in order to find a point close to the lattice, our sampler implicitly needs to know the lattice point.

Finding an Acceptable Output. Our solution to the previous problem is the following: rather than output samples \mathbf{A}_i and the closest lattice point to $\mathbf{A}_i \mathbf{s}$, what if we only output some limited information about that lattice point, rather than the whole thing? It turns out that if we write the ‘nearest’ lattice point in $\Lambda(\mathbf{B})$ in the form $\mathbf{B}\mathbf{u}$ (according to some rounding algorithm \mathcal{A} —we may not actually output the closest lattice point) and then output $\mathbf{u} \bmod 2$, we can actually prove security. We develop a variant of the LWE assumption that reduces from the standard LWE assumption that allows us to do this while maintaining the properties of unbounded (polynomial) samples and a polynomially-sized modulus.

⁴ Proving that an adversary cannot distinguish points close to the ‘rounding boundary’ of a lattice from uniformly random points without breaking some form of LWE seems like a very interesting open problem.

Informally, our assumption says that, for certain \mathbf{A}_i s picked so that $\mathbf{A}_i \mathbf{s}$ is close to $\Lambda(\mathbf{B})$, samples of the form

$$\left(\mathbf{A}_i, \lceil \mathbf{A}_i \mathbf{s} \rceil_{\Lambda(\mathbf{B})}^{\mathcal{A}} \pmod{2} \right)$$

are indistinguishable from random. Note that our process is parameterized by a lattice $\Lambda(\mathbf{B})$ and a rounding algorithm \mathcal{A} . We are obviously glossing over some important details here, but this is the basic idea (and the details are in the body of the paper).

2 Preliminaries

In this section we present some basic material common to many cryptographic papers. A reader familiar with general cryptography and particularly lattices and lattice cryptography can probably safely skip this section. We borrow elements of the presentation from [BGG+14, GPV08].

2.1 General Notation

For a random variable X we denote by $x \leftarrow X$ the process of sampling a value x from the distribution of X . If S is a finite set instead of a distribution, we denote by $x \leftarrow S$ the process of sampling a value x according to the uniform distribution over S .

A non-negative function $v(\lambda)$ is *negligible* if for every polynomial $p(\lambda)$ it holds that $v(\lambda) \leq \frac{1}{p(\lambda)}$ for all sufficiently large $\lambda \in \mathbb{N}$.

Statistical Distance: Let Ω be a finite domain, and let X and Y be random variables over Ω . We define the statistical distance between X and Y , denoted $SD(X, Y)$ in the following way:

$$SD(X, Y) = \frac{1}{2} \sum_{\omega \in \Omega} |\Pr[X = \omega] - \Pr[Y = \omega]|$$

We say that two variables X and Y are δ -close if $SD(X, Y) \leq \delta$. If we parameterize X and Y with the security parameter $\lambda \in \mathbb{N}$, we can say that two families of distributions X_λ and Y_λ are *statistically indistinguishable* or *statistically close* if $SD(X_\lambda, Y_\lambda)$ is negligible in λ .

Rounding. For an integer $p \leq q$, we define the modular “rounding” function

$$\lceil \cdot \rceil_p : \mathbb{Z}_q \rightarrow \mathbb{Z}_p \text{ that maps } x \rightarrow \lfloor (p/q) \cdot x \rfloor$$

and extend it coordinate-wise to matrices and vectors over \mathbb{Z}_q .

B-Bounded. Let $B = B(n) \in \mathbb{N}$ be a natural number. A family of distributions $\psi \in \{\psi_n\}_{n \in \mathbb{N}}$ is called B -bounded if $\Pr[\psi \in [-B, B]] = 1 - \text{negl}(n)$.

2.2 Lattice Background

Lattice Notation: let $q, n,$ and m be positive integers, and let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be a matrix. We let $\Lambda_q^\perp(\mathbf{A})$ denote the lattice spanned by all $\mathbf{x} \in \mathbb{Z}_q^m$ such that $\mathbf{A} \cdot \mathbf{x} = \mathbf{0} \pmod q$. For a vector $\mathbf{u} \in \mathbb{Z}_q^n$, we generalize this and let $\Lambda_q^\perp(\mathbf{A}, \mathbf{u})$ denote the set of all vectors such that $\mathbf{A} \cdot \mathbf{x} = \mathbf{u}$. Note that this is a coset of $\Lambda_q^\perp(\mathbf{A})$.

Discrete Gaussians. We borrow the elegant presentation style of [GPV08]. For any $s > 0$ define the Gaussian function on \mathbb{R}^n centered at \mathbf{c} with parameter \mathbf{s} :

$$\forall \mathbf{x} \in \mathbb{R}^n, \quad \rho_{\mathbf{s}, \mathbf{c}}(\mathbf{x}) = e^{-\pi \|\mathbf{x} - \mathbf{c}\|^2 / s^2}$$

We sometimes omit the subscripts s and c in the case that they are 1 and 0, respectively.

For any $\mathbf{c} \in \mathbb{R}^n$, real $s > 0$, and n -dimensional lattice Λ , we define the discrete Gaussian distribution over Λ as:

$$\forall \mathbf{x} \in \Lambda, \quad \mathcal{D}_{\Lambda, \mathbf{s}, \mathbf{c}}(\mathbf{x}) = \frac{\rho_{\mathbf{s}, \mathbf{c}}(\mathbf{x})}{\rho_{\mathbf{s}, \mathbf{c}}(\Lambda)}$$

Smoothing Parameter. In [MR04], Micciancio and Regev defined the smoothing parameter: for any n -dimensional lattice Λ and positive real ε , the *smoothing parameter* $\eta_\varepsilon(\Lambda)$ is the smallest real $s > 0$ such that $\rho_{1/s}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \varepsilon$.

2.3 LWE and Related Assumptions

Definition 1. Learning with Errors Problem (LWE): Consider integers n and q , some distribution ψ over \mathbb{Z}_q , and distributions \mathcal{K} and \mathcal{T} , both over \mathbb{Z}_q^n .

A $(q, n, \psi, \mathcal{K}, \mathcal{T})$ -LWE problem instance consists of access to an unspecified challenge oracle \mathcal{O}^{LWE} , being, either, a noisy pseudorandom sampler \mathcal{O}_s^{LWE} carrying some constant random secret key $\mathbf{s} \in \mathbb{Z}_q^n$ sampled from the distribution \mathcal{K} , or, a truly random sampler \mathcal{O}_s^{LWE} , whose behaviors are respectively as follows:

\mathcal{O}_s^{LWE} : Outputs samples of the form $(\mathbf{a}_i, \mathbf{a}_i \cdot \mathbf{s} + \delta_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{s} \in \mathbb{Z}_q^n$ is a persistent value invariant across invocations sampled by querying the distribution \mathcal{K} , $\delta_i \in \mathbb{Z}_q$ consists of a fresh sample from ψ , and $\mathbf{a}_i \in \mathbb{Z}_q^n$ is sampled at random from \mathcal{T} .

\mathcal{O}_s^{LWE} : Outputs samples of the form $(\mathbf{a}_i, \mathbf{r}_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{a}_i \in \mathbb{Z}_q^n$ is sampled at random from \mathcal{T} and \mathbf{r}_i is a uniform random sample from \mathbb{Z}_q .

The $(q, n, \psi, \mathcal{K}, \mathcal{T})$ -LWE problem allows repeated queries to the challenge oracle \mathcal{O}^{LWE} . We say that an algorithm \mathcal{A} decides the $(q, n, \psi, \mathcal{K}, \mathcal{T})$ -LWE problem if

$$\text{Adv}^{LWE}[\mathcal{A}] \stackrel{\text{def}}{=} |\Pr[\mathcal{A}^{\mathcal{O}_s^{LWE}} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_s^{LWE}} = 1]|$$

is non-negligible for a \mathbf{s} selected appropriately at random from \mathcal{K} .

Definition 2. Learning with Rounding Problem (LWR): Consider integers $n, p,$ and q such that $q \geq p,$ and distributions \mathcal{K} and $\mathcal{T},$ both over $\mathbb{Z}_q^n.$

A $(q, p, n, \mathcal{K}, \mathcal{T})$ -LWR problem instance consists of access to an unspecified challenge oracle $\mathcal{O}^{LWR},$ being, either, a noisy pseudorandom sampler \mathcal{O}_s^{LWR} carrying some constant random secret key $\mathbf{s} \in \mathbb{Z}_q^n$ sampled from the distribution $\mathcal{K},$ or, a truly random sampler $\mathcal{O}_s^{LWR},$ whose behaviors are respectively as follows:

$\mathcal{O}_s^{LWR}:$ Outputs samples of the form $(\mathbf{a}_i, \lceil \mathbf{a}_i \cdot \mathbf{s} \rceil_p) \in \mathbb{Z}_q^n \times \mathbb{Z}_p,$ where $\mathbf{s} \in \mathbb{Z}_q^n$ is a persistent value invariant across invocations sampled by querying the distribution \mathcal{K} and $\mathbf{a}_i \in \mathbb{Z}_q^n$ is sampled at random from $\mathcal{T}.$

$\mathcal{O}_s^{LWR}:$ Outputs samples of the form $(\mathbf{a}_i, \mathbf{r}_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_p,$ where $\mathbf{a}_i \in \mathbb{Z}_q^n$ is sampled at random from \mathcal{T} and \mathbf{r}_i is a uniform random sample from $\mathbb{Z}_p.$

The $(q, p, n, \mathcal{K}, \mathcal{T})$ -LWR problem allows repeated queries to the challenge oracle $\mathcal{O}^{LWR}.$ We say that an algorithm \mathcal{A} decides the $(q, p, n, \mathcal{K}, \mathcal{T})$ -LWR problem if

$$\text{Adv}^{LWR}[\mathcal{A}] \stackrel{\text{def}}{=} |\Pr[\mathcal{A}^{\mathcal{O}_s^{LWR}} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_s^{LWR}} = 1]|$$

is non-negligible for a \mathbf{s} selected appropriately at random from $\mathcal{K}.$

3 The Main Problem

We now are in a position to formally describe our new assumption. Before we begin with the actual problem, though, we need to formally define lattice rounding.

Lattice Rounding. We are going to overload the rounding function. Let $q, m,$ and n be integers with $m \geq n.$ For some deterministic rounding algorithm \mathcal{A} and lattice basis $\mathbf{B} \in \mathbb{Z}_q^{m \times n},$ we define the ‘‘rounding’’ function

$$\begin{aligned} \lceil \cdot \rceil_{\mathcal{A}(\mathbf{B})}^{\mathcal{A}} : \mathbb{Z}_q^m &\rightarrow \mathbb{Z}_q^n \text{ that maps } \mathbf{x} \rightarrow \mathbf{u} \\ &\text{such that } \mathcal{A}(\mathbf{x}) = \mathbf{B} \cdot \mathbf{u} \end{aligned}$$

Note that, while our rounding algorithm \mathcal{A} is deterministic, it doesn’t necessarily have to be perfect at rounding points to close lattice points. Since finding the (exact) closest lattice point is thought to be hard even when arbitrary ‘hints’ about the lattice are known, we cannot expect any rounding algorithm to always find the closest lattice points. In order to give a general definition, we only require that our rounding algorithm is deterministic.

However, we do note that, in order for our proof to hold, our rounding algorithm must round points within some error factor ϵ of a lattice point to the correct lattice point, but this ϵ can be substantially smaller than the shortest vector in the lattice. Examples of \mathcal{A} could be Babai’s nearest plane algorithm or Babai rounding with a ‘good’ lattice basis, for instance, although we use more complicated rounding algorithms in this work.

In addition, note that we output the coefficient vector \mathbf{u} instead of an actual lattice point like $\mathbf{B} \cdot \mathbf{u}$. While in principle we could output a lattice point, it will make the presentation of our results simpler if we define lattice rounding to output coefficient vectors instead of lattice points. This choice also seemingly makes lattice rounding easier to use, as most potential applications would prefer to have a uniformly random vector instead of a uniformly random lattice point as output.

3.1 Nearby Learning with Lattice Rounding Description

We next describe the details of our lattice rounding system.

Nearby Learning with Lattice Rounding

Input:

- Integers n, m , and q such that $m = O(n \log q)$
- A ‘noise’ distribution $\psi \in \mathbb{Z}_{2q}$
- Algorithm $GenTrap(1^n, 1^m, q) \rightarrow \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_{2q}^{\frac{m}{2} \times \frac{m}{2}}$ which outputs a parity check matrix $\mathbf{B} \in \mathbb{Z}_q^{m \times n}$ and a ‘trapdoor’ \mathbf{T}_B as described in the full paper.
- Algorithm $Invert(\mathbf{B}, \mathbf{T}_B, \mathbf{b} \in \mathbb{Z}_{2q}^m) \rightarrow \mathbb{Z}_{2q}^n$ which outputs the vector \mathbf{u} where $\mathbf{b} = \mathbf{B}\mathbf{u} + \mathbf{e}$ for some short vector $\mathbf{e} \in \mathbb{Z}_{2q}^m$ as described in the full paper.

Setup:

- Set $\mathbf{B}, \mathbf{T}_B \leftarrow GenTrap(1^n, 1^m, q)$.
- Set $\mathbf{s} \leftarrow \mathbb{Z}_{2q}^m$.

Output a Sample:

- Set $\mathbf{u}_i \leftarrow \mathbb{Z}_{2q}^n$.
- Set $\delta_i \in \mathbb{Z}_{2q}^m$ by concatenating m samples from ψ .
- Sample $\mathbf{A}_i \in \mathbb{Z}_{2q}^{m \times m}$ by choosing a random \mathbf{A}_i that satisfies $\mathbf{A}_i \mathbf{s} = \mathbf{B}\mathbf{u}_i + \delta_i$.
- **Output** $(\mathbf{A}_i, [\mathbf{A}_i \mathbf{s}]_{\Lambda(\mathbf{B})}^{Invert(\cdot)} \bmod 2) = (\mathbf{A}_i, \mathbf{u}_i \bmod 2)$.

Reconstruct a Sample Given \mathbf{A}_i :

- **Output** $(\mathbf{A}_i, [\mathbf{A}_i \mathbf{s}]_{\Lambda(\mathbf{B})}^{Invert(\cdot)} \bmod 2)$
-

Some Comments. We note that the algorithms $GenTrap$ and $Invert$ closely resemble those from [MP12]. Any standard lattice trapdoor algorithm will work here, but we chose to use this one for its ease of use and efficiency. $Invert$ is the function that takes the role of our abstract rounding algorithm \mathcal{A} , and $GenTrap$ gives us the information necessary to run $Invert$.

Our modulus $2q$ is an artifact of our proof technique, so it is not clear that this modulus is absolutely necessary for the problem to be hard. Let k be any constant integer. It is also worth pointing out that the problem can be easily modified to work with modulus kq and outputs of the form $(\mathbf{A}_i, [\mathbf{A}_i \mathbf{s}]_{\Lambda(\mathbf{B})}^{Invert(\cdot)} \bmod k)$, as we essentially just swap the 2 factor for some other constant k . However, we stick with the mod 2 case because generalizing to k doesn’t give us anything too novel from a theoretical perspective and outputting more randomness with each sample still does not make the problem practical.

A natural question to ask is why we need to generate \mathbf{B} with a trapdoor. For many applications of LWR (like PRFs, for instance) we need to recompute the output of a sample \mathbf{A}_i given only the sample \mathbf{A}_i and the key \mathbf{s} (and not the actual output). For basic LWR, this is trivial, but for NLWLR, this is difficult without a trapdoor. Our algorithm ‘Reconstruct a Sample Given \mathbf{A}_i ’ handles this functionality and needs to use the lattice trapdoor to work. We comment on this more in our security proof.

Additive Homomorphism. We also note that our outputs are ‘almost’ homomorphic—i.e., if the error is small enough, then:

$$\begin{aligned} & \lceil \mathbf{A}_i \mathbf{s} \rceil_{\Lambda(\mathbf{B})}^{\text{Invert}(\cdot)} \pmod 2 + \lceil \mathbf{A}_j \mathbf{s} \rceil_{\Lambda(\mathbf{B})}^{\text{Invert}(\cdot)} \pmod 2 = \\ & \lceil (\mathbf{A}_i + \mathbf{A}_j) \mathbf{s} \rceil_{\Lambda(\mathbf{B})}^{\text{Invert}(\cdot)} \pmod 2 \end{aligned}$$

This means that many standard cryptographic schemes that are built upon LWE can be built (very inefficiently) from the NLWLR assumption as well in a fairly straightforward manner. For instance, Regev-like encryption [Reg05] can be done fairly easily, and we briefly sketch how here: create a public key consisting of k samples of the form $(\mathbf{A}_i, \lceil \mathbf{A}_i \mathbf{s} \rceil_{\Lambda(\mathbf{B})}^{\text{Invert}(\cdot)} \pmod 2)$ and a secret key including \mathbf{s} and $\mathbf{T}_\mathbf{B}$.

To encrypt a bit vector $\mathbf{b} \in \mathbb{Z}_2^n$, pick a discrete Gaussian value c_i over \mathbb{Z} for every sample \mathbf{A}_i , and output the tuple

$$\left(\sum_{i=1}^k c_i \mathbf{A}_i \pmod{2q}, \sum_{i=1}^k c_i \lceil \mathbf{A}_i \mathbf{s} \rceil_{\Lambda(\mathbf{B})}^{\text{Invert}(\cdot)} + \mathbf{b} \pmod 2 \right)$$

Decryption is done in the natural Regev way. As long as the noise blow-up is controlled, then correctness follows. Uniformity over the choice of lattice point (i.e. $\sum_{i=1}^k c_i \mathbf{u}_i$) can be shown from the leftover hash lemma [BDK+11], and the leftover hash lemma over the integers [AGHS13] can be used to show that the noise term (i.e. $\sum_{i=1}^k c_i \boldsymbol{\delta}_i$) is distributed as a proper discrete ellipsoid, meaning that we end up with a well-formed, properly chosen sample \mathbf{A}_i and output as our sum. This will require an extremely large k to be used but we already said that this would be inefficient.

While we do not want to spend too much time going through inefficient cryptosystems (thus why we do not offer a formal treatment of the previous argument), we just want to illustrate that this new NLWLR assumption can, in fact, be used to build reasonably powerful cryptosystems.

Parameter Choices. We next put forth a set of parameter choices that offers easy instantiation and good theoretical hardness. Let m , n , and q be integers such that $m \geq 4n \log q$. Let the distribution $\mathcal{D}_{\mathbb{Z}, \sigma}$ be a discrete Gaussian distribution with $\sigma \leq \frac{q}{5m \log n}$ (so that the overall distribution over \mathbb{Z}^m is a discrete Gaussian $\mathcal{D}_{\mathbb{Z}^m, \sigma'}$ with parameter $\sigma' \leq \frac{q}{5\sqrt{m \log n}}$).

Existence of Efficient GenTrap and Invert. In the full version of the paper we prove that the algorithms *GenTrap* and *Invert* exist as defined and that *Invert* is deterministic. We prove in Lemma 2, that, for the choice of parameters mentioned above, the output of *GenTrap* is uniform and that the *Invert* algorithm is correct with all but negligible probability.

Reduction from Standard LWE. Also in Lemma 2 we show that our construction is at least as hard as standard LWE (uniform samples, uniform key) in dimension n over modulus q and with discrete Gaussian noise with parameter $\frac{\sigma}{3}$ for the parameters described above. If Q is the number of samples an adversary gets, we do lose a factor of Q in our security reduction, but this doesn't make an enormous difference theoretically. This means that, through the LWE reduction of [BLP+13], we have a polynomially-approximate reduction from worst-case lattice problems while still allowing an unbounded (polynomial) number of queries.

We note that our reduction allows for substantially more general parameters than the ones mentioned in this section.

4 Security Proof

In this section we prove the security of what we have called *Nearby Learning with Lattice Rounding*. Our security proof is actually rather straightforward, but has a couple of twists that can make it difficult to follow, so we offer a proof outline below to make it easier to digest. We start by going over an uncommon way to look at the standard LWE problem.

Unfortunately, we do not have space to present a full security proof in this format. Instead, we offer a proof outline which explains all of the intuition required to follow the proof. For the full security proof, please see the full version of the paper⁵.

4.1 Proof Outline

The core portion of our security proof has two main steps: first, we show that a particular nonuniform variant of LWE is hard. Then we show that anyone that can distinguish outputs of our nearby learning with lattice rounding oracle from random can break this nonuniform LWE variant. Once we have shown that our output is indistinguishable from random, we show that we can make the rounding mechanism work by appropriately sampling a random lattice and a trapdoor that are compatible with our scheme.

Reverse LWE. Our proof will become much easier to understand when viewed through lens of what we call 'reverse LWE'. Traditionally, LWE is thought of as the following procedure: given a fixed key \mathbf{s} , sample a random sample \mathbf{a}_i and a noise sample δ_i and output $(\mathbf{a}_i, \mathbf{a}_i^\top \mathbf{s} + \delta_i)$.

⁵ Available on eprint.

However, we can also generate LWE samples in the following way: suppose we are given a fixed key \mathbf{s} . First, pick some random value u over the output domain. Then, sample a value δ_i from the noise distribution, and choose a random \mathbf{a}_i subject to the constraint that $\mathbf{a}_i^\top \mathbf{s} = u - \delta_i$. It should be clear that this distribution is statistically close to that of the traditional LWE distribution, but what this variant of LWE gives us the power to do is to choose our outputs to be distributed in a non-uniform way if we like. Picking us (or, more precisely, vectors of us) from a distribution that is computationally indistinguishable from uniform rather than one that is statistically close to uniform is something that we implicitly exploit in our reduction.

A New Nonuniform LWE. Our reduction starts by showing that the following is a hard (nonuniform) instance of LWE: samples of the form $(\mathbf{a}_i, \mathbf{a}_i^\top (2\mathbf{s}) + \delta_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_{2q}$, where the operations are done modulo \mathbb{Z}_{2q} but the $\mathbf{a}_i \mathbf{s}$ and \mathbf{s} are sampled uniformly modulo q . The proof goes approximately as follows: we start with a typical LWE instance over \mathbb{Z}_q . Imagine we have samples of the form $(\mathbf{a}_i, \mathbf{a}_i^\top \mathbf{s} + \delta_i)$. Suppose we multiply the second term by two and ‘lift’ everything up to \mathbb{Z}_{2q} : we now have samples of the form $(\mathbf{a}_i, \mathbf{a}_i^\top (2\mathbf{s}) + 2\delta_i)$. Note that the second term of the tuple is, by the LWE assumption, indistinguishable over the higher-order bits (the lowest-order bit will be zero since the modulus is a multiple of two). We can then add in additional noise to make the $2\mathbf{s} + 2\delta$ term look uniform since the only nonuniformity is in the lowest-order bit.

This gives us LWE samples of the form $(\mathbf{a}_i, \mathbf{a}_i^\top (2\mathbf{s}) + \delta_i)$, where both \mathbf{a}_i and \mathbf{s} are uniform over \mathbb{Z}_q^n rather than \mathbb{Z}_{2q}^n that are still indistinguishable from random (for any attentive readers, it is straightforward to show that the ‘random’ part of the LWE reduction holds as well).

Moving to Rounding. We now have an LWE problem where we can manipulate the lowest-order bits of the key. Given samples of the form $(\mathbf{a}_i, \mathbf{a}_i^\top (2\mathbf{s}) + \delta_i)$, we can sample a term $\mathbf{t} \in \mathbb{Z}_2^n$ and add in $\mathbf{a}_i^\top \mathbf{t}$ to the second term of the tuple, which gives us a uniform key over \mathbb{Z}_{2q}^n (but we have knowledge of the lowest-order bits of the key!).

Suppose we now want to sample a single query from our rounding oracle. We can take m LWE samples in the above form and concatenate them as rows into a matrix which we will conveniently call $\mathbf{B} \in \mathbb{Z}_q^n$. We have something of the form $(\mathbf{B}, \mathbf{B}\mathbf{s} + \boldsymbol{\delta})$ where we know the lowest-order bits of \mathbf{s} (in other words, $\mathbf{s} \pmod 2$).

Given a random vector $\mathbf{t} \in \mathbb{Z}_{2q}^m$, we can set a matrix $\mathbf{A}_i \in \mathbb{Z}_{2q}^m$ such that $\mathbf{A}_i \mathbf{t} = \mathbf{B}\mathbf{s} - \boldsymbol{\delta}$. Thus we can output a sample $(\mathbf{A}_i, \mathbf{s} \pmod 2)$ and have it be a valid output for our nearby learning with lattice rounding oracle since $\mathbf{B}\mathbf{s}$ is the closest point in $\Lambda(\mathbf{B})$ to $\mathbf{A}_i \mathbf{t}$. It is also straightforward to show that \mathbf{A}_i is distributed uniformly at random (and independent of $\mathbf{s} \pmod 2$) if the original LWE samples were random.

Unfortunately, given that the secret in our reduction is a close lattice point (which only gives us one query) rather than many queries (as a LWE secret might), we are forced to use a hybrid argument over all of the adversary’s queries

Q . This gives us a $\frac{1}{Q}$ -security loss in our reduction, but we still can output an unbounded number of queries because Q must be polynomial at the end of the day. We defer the details to the formal proof.

Actually Rounding. While the above argument explains why our output is indistinguishable from random, we would like to be able to actually round as well (for a valid \mathbf{A}_i , any entity with the requisite secrets \mathbf{s} and \mathbf{T}_B should be able to produce a valid output) instead of just simulating rounded output. This is essential for sort of any application where multiple parties need to statelessly produce the same outputs given repeated samples \mathbf{A}_i as input⁶. The most basic way to do this would be to sample our lattice basis \mathbf{B} with a trapdoor. Recall that, in [MP12] the authors provide algorithms for sampling a random-looking parity check matrix $\mathbf{B} \in \mathbb{Z}_q^{m \times n}$ and a corresponding trapdoor \mathbf{T}_B such that, given the trapdoor \mathbf{T}_B and a sample of the form $\mathbf{B}\mathbf{u} + \boldsymbol{\delta} \in \mathbb{Z}_q^m$ (for some $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ and some noise vector $\boldsymbol{\delta} \in \mathbb{Z}_q^n$) it is possible to efficiently determine \mathbf{u} with high probability.

Unfortunately, we want an instance of this algorithm where \mathbf{B} still lives in \mathbb{Z}_q but the output samples and the secret \mathbf{u} live in \mathbb{Z}_{2q} . As an astute reader might notice, this is slightly annoying to go through all of the details, but conceptually simple. To illustrate, suppose we have a collection of short vectors \mathbf{x} such that $\mathbf{B}\mathbf{x} = \mathbf{0} \pmod q$, giving us a form of trapdoor for \mathbf{B} . Then $\mathbf{B}(2\mathbf{x}) = \mathbf{0} \pmod{2q}$, which gives us a slightly worse form of trapdoor. In sum, getting this trapdoor sampler to work for a different modulus of output samples is fairly straightforward, if a bit tedious. We slightly modify the algorithms from [MP12] in order to do this easily and efficiently.

4.2 Formal Security Statements

We now formally define the Nearby Learning with Lattice Rounding Assumption. We note that the output of this distribution (in the ‘real case’) is identical to that of the output described in Sect. 3.1, as required by definition.

Definition 3. *Nearby Learning with Lattice Rounding (NLWLR):* Consider integers m, n , and q , some B -bounded distribution ψ over \mathbb{Z}_{2q} , and a distribution \mathcal{T} over \mathbb{Z}_q^n .

Let the matrix $\mathbf{B} \in \mathbb{Z}_q^{m \times n}$ be distributed such that each row is a fresh sample from \mathcal{T} . We let $\Lambda(\mathbf{B})$ denote the $2q$ -ary lattice with basis \mathbf{B} .

A $(q, n, \psi, \mathcal{K}, \mathbf{B})$ -NLWLR problem instance consists of access to the public matrix \mathbf{B} and an unspecified challenge oracle \mathcal{O}^{NLWLR} , being, either, a noisy pseudorandom sampler \mathcal{O}_s^{NLWLR} carrying some constant random secret key $\mathbf{s} \in \mathbb{Z}_{2q}^n$ sampled from the distribution \mathcal{K} , or, a truly random sampler \mathcal{O}_s^{NLWLR} , whose behaviors are respectively as follows:

\mathcal{O}_s^{NLWLR} : Outputs samples of the form $(\mathbf{A}_i, \mathbf{z}_i) \in \mathbb{Z}_{2q}^{m \times m} \times \mathbb{Z}_2^m$, where the terms are sampled in the following way: first, recall that $\mathbf{s} \in \mathbb{Z}_q^n$ is a persistent

⁶ Like PRFs, for instance. We discuss PRFs in the conclusion.

value invariant across invocations sampled by querying the distribution \mathcal{K} . Let the value \mathbf{u}_i be sampled randomly from \mathcal{U}_{2q}^n , and let $\delta_i \in \mathbb{Z}_{2q}^m$ be sampled by selecting m independent fresh samples from ψ and concatenating them together to form a vector. To output \mathbf{A}_i , we use Gaussian elimination to solve for the matrix \mathbf{A}_i that satisfies the following equation:

$$\mathbf{A}_i \cdot \mathbf{s} = \mathbf{B} \cdot \mathbf{u}_i + \delta_i$$

We then set $\mathbf{z}_i = \mathbf{u}_i \pmod 2$ and output that as well.

\mathcal{O}_s^{NLWLR} : Outputs samples of the form $(\mathbf{A}_i, \mathbf{z}_i) \in \mathbb{Z}_{2q}^{m \times m} \times \mathbb{Z}_2^m$, where $\mathbf{A}_i \in \mathbb{Z}_q^{m \times n}$ is sampled uniformly at random from $\mathbb{Z}_q^{m \times m}$ and \mathbf{z}_i is a randomly sampled vector in \mathbb{Z}_2^n .

The $(q, n, \psi, \mathcal{K}, \mathbf{B})$ -NLWLR problem allows repeated queries to the challenge oracle \mathcal{O}^{NLWLR} . We say that an algorithm \mathcal{A} decides the $(q, n, \psi, \mathcal{K}, \mathbf{B})$ -NLWLR problem if

$$\text{Adv}^{NLWLR}[\mathcal{A}] \stackrel{\text{def}}{=} |\Pr[\mathcal{A}^{\mathcal{O}_s^{NLWLR}} = 1] - \Pr[\mathcal{A}^{\mathcal{O}_s^{NLWLR}} = 1]|$$

is non-negligible for a \mathbf{s} selected appropriately at random from \mathcal{K} .

Lemma 1. Consider integers m, n , and q , some B -bounded distribution ψ over \mathbb{Z}_{2q} , and a distribution \mathcal{T} over \mathbb{Z}_q^n .

Let the matrix $\mathbf{B} \in \mathbb{Z}_q^{m \times n}$ be distributed such that each row is a fresh sample from \mathcal{T} . We let $\Lambda(\mathbf{B})$ denote the $2q$ -ary lattice with basis \mathbf{B} .

Let Q be the number of queries made to an NLWLR oracle. Any adversary that can solve the $(q, n, \psi, \mathcal{U}_q^n, \mathbf{B})$ -NLWLR problem with advantage ε can be used to solve the $(2q, n, \psi, 2 \cdot \mathcal{U}_q^n, \mathcal{T})$ -LWE problem with advantage at least $\frac{1}{Q}\varepsilon$.

Proof. Please refer to the full version of this paper.

Generating a Trapdoor. In the full version of the paper, we formally show that there are suitable mechanisms for us to realize the *GenTrap* and *Invert* functions needed for the nearby learning with lattice rounding assumption to work. As we mentioned in the proof overview, this is really quite straightforward, but still takes a bit of digging through the details in order to build a convincing proof. Our construction of the *GenTrap* and *Invert* functions are almost identical to (and the notation is borrowed from) those of [MP12], but in theory any kind of trapdoor generation could be used. We defer the presentation and proof of these algorithms to the full version of the paper.

Main Lemmas. We now formally a lemma that we prove in the full version of the paper. We offer more general formulations in the full version as well.

Lemma 2. Let m, n , and q be integers such that $m \geq 4n \log q$. Let $\bar{m} = w = 2n \log q$, and let the distribution \mathcal{D} output matrices $\mathbf{R} \in \mathbb{Z}^{w \times \bar{m}}$ such that each entry \mathbf{R}_{ij} is equal to one with probability $\frac{1}{4}$, -1 with probability $\frac{1}{4}$, and

zero otherwise. Finally, let the distribution $\psi = \mathcal{D}_{\mathbb{Z},\sigma}$ be a discrete Gaussian distribution with $\sigma \leq \frac{q}{5m \log n}$. Let $\mathbf{B} \in \mathbb{Z}_q^{m \times n}$ be the output of *GenTrap*. We also require that $\sigma \geq 3\eta_\varepsilon(\mathbb{Z}^m)$.

Then there exist *GenTrap* and *Invert* algorithms that are correct with all but negligible probability. In addition, any adversary that can solve the $(q, n, \mathcal{D}_{\mathbb{Z},\sigma}, \mathcal{U}_q^n, \mathbf{B})$ -NLWLR problem with advantage ε can be used to solve the $(q, n, \mathcal{D}_{\mathbb{Z},\frac{\sigma}{3}}, \mathcal{U}_q^n, \mathcal{U}_q^n)$ -LWE problem with advantage $\frac{1}{Q}\varepsilon$.

5 Conclusion and Open Problems

In this work, we developed a deterministic variant of LWE with polynomial modulus and unbounded samples and showed that it is as hard as standard LWE. To our knowledge, this is the first such construction. Below we summarize our results in a table. Let c be a constant⁷, $\gamma \geq 1$, let B be the B -bound for the noise distribution of the LWE instance that the LWR instance reduces from, and let κ be a (negligible) security parameter. We have:

Work	Unbounded samples (w)	Modulus (q)	Advantage change ($\varepsilon \rightarrow \varepsilon'$)
[BPR12]	Yes	$Bp\kappa^{\omega(1)}$	$\varepsilon - \mathbf{negl}(\kappa)$
[AKPW13]	No	$\gamma Bwp\kappa$	$\varepsilon/(2dw)$
[BGM+16]	No	Bwp	$(\varepsilon/qw)^2$
[BLL+15]	No	Bwp	$(\varepsilon/qw)^2$
[AA16] (1)	No	$Bwp\kappa$	$\varepsilon(wB)^{-c}$
[AA16] (2)	No	$Bwp\kappa$	$\varepsilon(w)^{-c}$
This	Yes	$O(B\sqrt{n \log q}) \cdot \omega(\sqrt{\log n})$	$\varepsilon(w)^{-1}$

Work	Dimension change ($d \rightarrow d'$)	Straightforward rounding	Uniform samples
[BPR12]	d	Yes	Yes
[AKPW13]	$d \log(\gamma)/\log q$	Yes	Yes
[BGM+16]	$d/\log_\rho q$	Yes	Yes
[BLL+15]	d	Yes	Yes
[AA16] (1)	d	Yes	Yes
[AA16] (2)	$d - c$	Yes	Yes
This	$d/O(\log q)$	No	No

⁷ For typical choices of parameters.

While our construction does offer unbounded samples and a polynomial modulus (and thus a polynomial approximation factor to worst-case lattice problems), it has some rather large drawbacks as well. The fact that the distribution of the samples \mathbf{A}_i is not uniform makes this new assumption much more difficult to use in practice. Additionally, rounding to a lattice rather than to the nearest multiple of some integer p means that we lose most (if not all) of the efficiency advantages of rounding when compared to regular LWE.

It is our hope that future research can be done to eliminate these steps or to provide evidence as to why they are essential to the security of learning with rounding over a polynomial modulus with unbounded samples.

On Building PRFs. Given that we are building a deterministic variant of LWE that supports an unbounded number of queries, a natural question to ask is whether we can build fully parallelizable PRFs using the NLWLR assumption with a polynomial modulus q . Unfortunately, this still seems like it will require a substantial amount of new ideas. Note that even if we could prove that the standard form of LWR was exactly as hard as LWE, then constructing a lattice-based PRF with polynomial modulus would still require new ideas. This is because all of the known parallelizable lattice-based PRFs involve at least some form of subset product LWE. In other words, these PRFs require things of the form

$$\prod_{i=1}^{\ell} \mathbf{A}_{i,b_i} \mathbf{s} + \boldsymbol{\delta}_j$$

for ‘random’ matrices \mathbf{A}_i , a secret key \mathbf{s} , input bits b_i , and fresh noise samples $\boldsymbol{\delta}_j$ to be hard. While the clever construction in [BP14] attempts to minimize the noise blowup of these subset products, even it must have these subset products in some (lesser) depth to maintain a healthy amount of parallelizability.

We consider proving the hardness of this subset product LWE problem for a polynomial modulus (and reducing the hardness of it to a polynomial value for the LWE security reduction parameter α) or providing evidence why it cannot be easily reduced to be an important open problem in this area.

Acknowledgements. We would like to thank Arnab Roy, Avradip Mandal, Sam Kim, David Wu, and Mike Hamburg for discussing lattices with us. We also want to thank the anonymous reviewers of the PQCrypto committee for their useful comments and suggestions on the paper and, in particular, anonymous reviewer #2 for their extremely helpful and detailed review.

References

- [AA16] Alperin-Sheriff, J., Apon, D.: Dimension-preserving reductions from LWE to LWR. IACR Cryptology ePrint Archive, 2016:589 (2016)
- [AGHS13] Agrawal, S., Gentry, C., Halevi, S., Sahai, A.: Discrete Gaussian leftover hash lemma over infinite domains. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8269, pp. 97–116. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42033-7_6

- [AKPW13] Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited - new reduction, properties and applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 57–74. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_4
- [BCD+16] Bos, J.W., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: take off the ring! Practical, quantum-secure key exchange from LWE. In: ACM CCS 2016: 23rd Conference on Computer and Communications Security, pp. 1006–1018. ACM Press (2016)
- [BDK+11] Barak, B., Dodis, Y., Krawczyk, H., Pereira, O., Pietrzak, K., Standaert, F.-X., Yu, Y.: Leftover hash lemma, revisited. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_1
- [BDK+17] 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 ePrint Archive, 2017:634 (2017)
- [BFP+15] Banerjee, A., Fuchsbauer, G., Peikert, C., Pietrzak, K., Stevens, S.: Key-homomorphic constrained pseudorandom functions. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 31–60. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_2
- [BGG+14] Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 533–556. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_30
- [BGM+16] Bogdanov, A., Guo, S., Masny, D., Richelson, S., Rosen, A.: On the hardness of learning with rounding over small modulus. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 209–224. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_9
- [BKM17] Boneh, D., Kim, S., Montgomery, H.: Private puncturable PRFs from standard lattice assumptions. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 415–445. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_15
- [BLL+15] Bai, S., Langlois, A., Lepoint, T., Stehlé, D., Steinfeld, R.: Improved security proofs in lattice-based cryptography: using the rényi divergence rather than the statistical distance. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 3–24. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_1
- [BLMR13] Boneh, D., Lewi, K., Montgomery, H., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_23
- [BLP+13] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th Annual ACM Symposium on Theory of Computing, Palo Alto, CA, USA, 1–4 June 2013, pp. 575–584. ACM Press (2013)
- [BP14] Banerjee, A., Peikert, C.: New and improved key-homomorphic pseudorandom functions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 353–370. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_20

- [BPR12] Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 719–737. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_42
- [BV11] Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd Annual Symposium on Foundations of Computer Science, Palm Springs, California, USA, 22–25 October 2011, pp. 97–106. IEEE Computer Society Press (2011)
- [CC17] Canetti, R., Chen, Y.: Constraint-hiding constrained PRFs for NC^1 from LWE. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 446–476. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_16
- [CKLS16] Cheon, J.H., Kim, D., Lee, J., Song, Y.: Lizard: cut off the tail! practical post-quantum public-key encryption from LWE and LWR (2016). <http://eprint.iacr.org/2016/1126>
- [DFH+16] Dziembowski, S., Faust, S., Herold, G., Journault, A., Masny, D., Standdaert, F.-X.: Towards sound fresh re-keying with hard (physical) learning problems. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 272–301. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_10
- [GGM84] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th Annual Symposium on Foundations of Computer Science, Singer Island, Florida, 24–26 October 1984, pp. 464–479. IEEE Computer Society Press (1984)
- [GPV08] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, 17–20 May 2008, pp. 197–206. ACM Press (2008)
- [MP12] Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
- [MR04] Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. In: 45th Annual Symposium on Foundations of Computer Science, Rome, Italy, 17–19 October 2004, pp. 372–381. IEEE Computer Society Press (2004)
- [NR06] Nguyen, P.Q., Regev, O.: Learning a parallelepiped: cryptanalysis of GGH and NTRU signatures. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 271–288. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_17
- [Pei14] Peikert, C.: Lattice cryptography for the internet. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 197–219. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_12
- [Reg05] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, USA, 22–24 May 2005, pp. 84–93. ACM Press (2005)
- [RRVV14] Roy, S.S., Reparaz, O., Vercauteren, F., Verbauwhede, I.: Compact and side channel secure discrete Gaussian sampling. Cryptology ePrint Archive, Report 2014/591 (2014). <http://eprint.iacr.org/2014/591>



Lattice-Based Signcryption Without Random Oracles

Shingo Sato^{1(✉)} and Junji Shikata^{1,2}

¹ Graduate School of Environment and Information Sciences,
Yokohama National University, Yokohama, Japan
sato-shingo-cz@ynu.jp

² Institute of Advanced Sciences, Yokohama National University, Yokohama, Japan

Abstract. Signcryption is a scheme that achieves both functionalities of public key encryption and digital signatures, and hence it is an important and fundamental protocol in cryptography. On the other hand, it is interesting to efficiently construct a signcryption scheme based on lattice-based problems, since lattice-based construction is expected to have resistance against quantum computing. The contribution of this paper is to construct an efficient lattice-based signcryption satisfying strong security without random oracles. We propose such a construction based on the problems of the learning with errors (LWE) and small integer solution (SIS). The public-key size and ciphertext size in our construction are shorter than any other schemes, and there is no disadvantage for ours in other parameters compared to other ones in terms of public/secret-key and ciphertext sizes.

Keywords: Signcryption · Lattice problems · LWE · SIS

1 Introduction

1.1 Background

Cryptographic technologies are used in communication for confidentiality and/or authenticity of information. Public-key encryption (PKE) and digital signature (DS) are especially important and elemental in cryptographic technologies. PKE enables us to communicate securely without sharing secret information in advance. On the other hand, DS is cryptographic technology that enables us to authenticate the composer and integrity of digital data. Namely, PKE and DS ensure confidentiality and authenticity of information, respectively. In contrast, signcryption is cryptographic technology that achieves both confidentiality and integrity, that is, both the functions of PKE and DS. Signcryption is important, since it realizes a secure channel from an insecure channel such as the Internet.

The notion of signcryption was introduced by Zheng [36]. In the model of signcryption, there are two kinds of setting, the *two-user setting* and *multi-user setting*. The two-user setting is a simple model of signcryption in which there

is a single sender and a single receiver. In contrast, the multi-user setting is the model where there are multiple senders and receivers. It is important to realize signcryption in the multi-user setting, since it is a realistic model and the security of two-user setting does not imply that of the multi-user setting. Furthermore, there are two kinds of security for signcryption, the *insider security* and *outsider security*. In the outsider security, an external adversary only knows public information (i.e., public parameters and public-keys of entities). On the other hand, in the insider security, an internal adversary can know some private-keys. Note that the insider security is stronger, and hence it is sufficient and reasonable to consider the insider security. The concrete differences between these models are described in Sect. 3. The strongest security definition, which consists of strong insider confidentiality and strong insider integrity in the multi-user setting, was first formalized by Libert and Quisquater [18]. In this paper, as IND-CCA security and sUF-CMA security in this security model, we call multi-user indistinguishability against insider chosen ciphertext attack (MU-IND-iCCA), and multi-user strong unforgeability against insider chosen message attack (MU-sUF-iCMA), respectively. Currently, there are several constructions known for signcryption schemes satisfying the strongest security, [4, 18, 22] in the random oracle model, and [11, 22, 25, 32] in the standard model (i.e., without random oracles). The construction in [32] is a direct construction, and constructions in [11, 22, 25] are generic constructions. Note that [22] requires a key registration, and it is desirable to construct signcryption without the key registration. In this sense, we focus on the generic constructions in [11, 25] in this paper.

On the other hand, researchers have recently paid much attentions to lattice-based cryptography, since it is expected that lattice problems can provide security against quantum computers and can realize advanced functionalities. As promising and interesting problems related with lattices, we can consider the problems of *learning with errors* (LWE) and *small integer solution* (SIS) which are focused on recently in constructions of lattice-based cryptography. So far, various and important constructions of cryptographic schemes have been proposed based on the problems: PKE in the standard model [19, 28–30]; and DS in the standard model [6, 8–10, 13, 21, 31, 35] and in the random oracle model [12, 14, 20]. In addition, there are constructions of key encapsulation mechanism (KEM) [10, 26], identity-based encryption [1, 9, 10, 14, 33, 35], oblivious transfer [28], and commitment [16]. Furthermore, there are constructions of hash functions such as collision-resistant hash functions [15, 24] and chameleon hash functions [10].

1.2 Our Contribution

The purpose of this paper is to construct an efficient latticed-based signcryption satisfying both MU-IND-iCCA and MU-sUF-iCMA security without random oracles¹. Specifically, our contribution of this paper is as follows:

¹ Although a construction of lattice-based signcryption without random oracles was proposed in [34], we confirmed that this construction didn't meet MU-sUF-iCMA security.

1. In Sect. 4.1, we propose a construction of signcryption without random oracles based on lattice problems. Although it is pointed out in [3, 22] that a trivial construction of signcryption by combining IND-CCA secure PKE and sUF-CMA secure DS based on the sign-then-encrypt paradigm cannot achieve MU-sUF-iCMA security, we can show that our construction based on the sign-then-encrypt paradigm can achieve both MU-IND-iCCA and MU-sUF-iCMA security. Namely, our lattice-based construction of signcryption meets both MU-IND-iCCA and MU-sUF-iCMA security under the LWE and SIS assumptions.
2. To improve efficiency of the lattice-based signcryption above, in Sect. 4.2, we propose a lattice-based hybrid signcryption obtained by combining our lattice-based signcryption and a DEM meeting indistinguishability against one-time attack (IND-OT) security. We show that this construction also achieves both of MU-IND-iCCA and MU-sUF-iCMA security under the LWE and SIS assumptions.
3. In Sect. 5, we compare efficiency of our construction in Sect. 4.2 with other lattice-based ones in terms of key-sizes and ciphertext-size. We note that there is no direct construction of signcryption based on lattice problems, though we can obtain several lattice-based constructions of signcryption by applying lattice-based primitives to the existing generic constructions [11, 25]. As a result, we show that our construction can achieve shortest ciphertext-size and there is no disadvantage in ours in other parameters.

2 Preliminaries

In this paper, we use the following notation. If we write $(y_1, y_2, \dots, y_m) \leftarrow A(x_1, x_2, \dots, x_n)$ for an algorithm A having n inputs and m outputs, it means to input x_1, x_2, \dots, x_n into A and to get the resulting output y_1, y_2, \dots, y_m . If A is a probabilistic algorithm, we write $(y_1, y_2, \dots, y_m) \leftarrow A(r; x_1, x_2, \dots, x_n)$, where r is a random value used in A . If x is a string, then $|x|$ denotes its bit-length. We denote a concatenation of x and y by $x||y$. If we write negligible ϵ in n (or denoted by $\epsilon = \text{negl}(n)$), it means a function $\epsilon : \mathbb{N} \rightarrow [0, 1]$ where $\epsilon(n) < 1/g(n)$ for any polynomial g and sufficiently large n . Furthermore, probabilistic polynomial time is abbreviated as PPT in this paper.

In this section, we describe definitions about lattices and related computational problems. For a positive integer n , $[n] := \{1, 2, \dots, n\}$. In this paper, vectors are assumed to be in column form. For a vector \mathbf{x} , the row vector is written as \mathbf{x}^T . As a norm of a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$, the Euclidean norm is denoted by $\|\mathbf{x}\|$.

Let X and Y be two random variables in a finite set Ω . The statistical distance of X and Y is defined as $\Delta(X, Y) = \frac{1}{2} \sum_{w \in \Omega} |\Pr[X = w] - \Pr[Y = w]|$. In addition, we say that X is ϵ -close to Y , if $\Delta(X, Y) \leq \epsilon$.

Furthermore, for completeness, we give the models and security of data encapsulation mechanisms (DEM) in Appendix A which will be considered in this paper.

2.1 Definitions for Lattices

An n -dimensional lattice Λ , which is generated by linearly independent vectors $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$, is defined as $\Lambda = \mathcal{L}(\mathbf{B}) = \{\sum_{i \in [n]} c_i \mathbf{b}_i : c_i \in \mathbb{Z}\}$.

We define a discrete Gaussian distribution. For any real number $\alpha > 0$, we define the Gaussian function $\rho_\alpha(\mathbf{x}) = \exp(-\pi\|\mathbf{x}\|^2/\alpha^2)$ for $\mathbf{x} \in \mathbb{R}^n$. Let $\rho_\alpha(\Lambda) := \sum_{\mathbf{x} \in \Lambda} \rho_\alpha(\mathbf{x})$ for an n -dimensional lattice Λ . The discrete Gaussian distribution for an n -dimensional lattice Λ is defined as follows: $\forall \mathbf{x} \in \Lambda$, we define $D_{\Lambda, \alpha}(\mathbf{x}) = \frac{\rho_\alpha(\mathbf{x})}{\rho_\alpha(\Lambda)}$.

A lattice-based collision-resistant hash function f (cf. [24]) is defined as follows: For a parameter $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and an input $\mathbf{x} \leftarrow D_{\mathbb{Z}, \delta}^m$ where $\delta(\ll q)$ is a small real number, $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q$. If $\|\mathbf{x}\| > \delta\sqrt{m}$, we can use the Merkle-Damgård construction such as a construction in [16].

For integers $n \geq 1, q \geq 2, m \geq 2$, and for a given parity-check matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we define the following m -dimensional lattices:

$$\begin{aligned} \Lambda_q(\mathbf{A}) &:= \{\mathbf{e} \in \mathbb{Z}^m : \exists \mathbf{s} \in \mathbb{Z}_q^n, \mathbf{A}^T \mathbf{s} = \mathbf{e} \bmod q\}, \\ \Lambda_q^\perp(\mathbf{A}) &:= \{\mathbf{e} \in \mathbb{Z}^m : \mathbf{A}\mathbf{e} = \mathbf{0} \bmod q\}, \\ \Lambda_q^u(\mathbf{A}) &:= \{\mathbf{e} \in \mathbb{Z}^m : \mathbf{A}\mathbf{e} = \mathbf{u} \bmod q\}. \end{aligned}$$

There are several computational problems related to the families of lattices above. Of those, we focus on the small integer solution (SIS) problem and the learning with errors (LWE) problem in this paper, which are recently paid much attentions in the lattice-based cryptography. These problems are stated in the following.

Definition 1 (SIS $_{q, \beta}$). *For an integer q and a real number β , given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, find an integer vector $\mathbf{e} \in \mathbb{Z}^m \setminus \{\mathbf{0}\}$ such that $\mathbf{e} \in \Lambda_q^\perp(\mathbf{A})$ and $\|\mathbf{e}\| \leq \beta$.*

Let $\mathbb{T} = \mathbb{R}/\mathbb{Z}$. Let D_α be the Gaussian probability distribution over \mathbb{R} with mean 0 and standard deviation $\alpha > 0$. For a vector $\mathbf{s} \in \mathbb{Z}_q^n$ and the Gaussian distribution D_α , let $A_{\mathbf{s}, \alpha}$ be the distribution on $\mathbb{Z}_q^n \times \mathbb{T}$ defined by the following: choose a vector $\mathbf{a} \xleftarrow{U} \mathbb{Z}_q^n$ and an error $e \leftarrow D_\alpha$, and then output $(\mathbf{a}, \mathbf{s}^T \mathbf{a}/q + e \bmod 1)$.

Definition 2 (LWE $_{q, \alpha}$). *For an integer q and a real number α , distinguish (with non-negligible probability), with oracle access to any desired $m = \text{poly}(n)$ samples, the distribution $A_{\mathbf{s}, \alpha}$ for $\mathbf{s} \in \mathbb{Z}_q^n$ from the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{T}$.*

2.2 Algorithms with Trapdoors

We describe lattice-based adaptive trapdoor functions. Micciancio and Peikert [23] introduced the following algorithms with trapdoors.

Proposition 1 ([23]). *There is an efficient randomized algorithm $\text{GenTrap}(1^n, 1^m, q)$ that, given any integers $n \geq 1, q \geq 2$ and sufficiently large $m = O(n \log n)$, outputs a parity-check matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a trapdoor \mathbf{T} such that the distribution of $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is $\text{negl}(n)$ -close to the uniform distribution. Moreover, there are efficient algorithms, denoted by Invert and SampleD , which do the following with overwhelming probability over all random choices:*

- For given \mathbf{T} , \mathbf{A} , and $\mathbf{b}^T = \mathbf{s}^T \mathbf{A} + \mathbf{e}^T$ as input, where $\mathbf{s} \in \mathbb{Z}_q^n$ and either $\|\mathbf{e}\| < q/O(\sqrt{n \log q})$ or $\mathbf{e} \leftarrow D_{\mathbb{Z}^m, \alpha q}$ for $1/\alpha \geq \sqrt{n \log q} \cdot \omega(\sqrt{\log n})$, the deterministic algorithm $\text{Invert}(\mathbf{T}, \mathbf{A}, \mathbf{b})$ outputs \mathbf{s} and \mathbf{e} .
- For given \mathbf{T} , \mathbf{A} , $\mathbf{u} \in \mathbb{Z}_q^n$, and large $s = O(\sqrt{n \log q})$ as input, the randomized algorithm $\text{SampleD}(\mathbf{T}, \mathbf{A}, \mathbf{u}, s)$ outputs samples from a distribution $\text{negl}(n)$ -close to $D_{\Lambda_q^u, s \cdot \omega(\sqrt{\log n})}$.

A matrix \mathbf{A} and the trapdoor \mathbf{T} , which meet the conditions in Proposition 1, are generated by the technique in Theorem 4.1 of [23] using a primitive matrix \mathbf{G} as follows: For a positive integer $k = \lceil \log q \rceil$ and a vector

$$\mathbf{g}^T = [1, 2, 2^2, \dots, 2^{k-1}] \in \mathbb{Z}_q^{1 \times k}, \mathbf{G} \text{ is defined by } \mathbf{G} := \begin{bmatrix} \mathbf{g}^T & 0 \\ & \ddots \\ 0 & \mathbf{g}^T \end{bmatrix} \in \mathbb{Z}_q^{n \times nk}.$$

The GenTrap algorithm in Proposition 1 takes a matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times m}$ and an invertible matrix $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ as input, and then chooses \mathbf{T} from a distribution D over $\mathbb{Z}^{m \times w}$, computes a matrix $\mathbf{A} = \lceil \bar{\mathbf{A}} \mathbf{H} \mathbf{G} - \bar{\mathbf{A}} \mathbf{T} \rceil$, and then outputs the parity-check matrix \mathbf{A} and the trapdoor \mathbf{T} . \mathbf{H} is a tag of the trapdoor.

3 Signcryption: Model and Security

In this section, we describe the model and security about signcryption.

Definition 3 (Signcryption). *A signcryption scheme SCS consists of a five-tuple of polynomial-time algorithms $\text{SCS} = (\text{Setup}, \text{KeyGen}_R, \text{KeyGen}_S, \text{SC}, \text{USC})$ as follows: Let \mathcal{MSP} be a message-space.*

- $\text{Setup}(1^k)$: Setup is a randomised setup algorithm that on input a security parameter k , outputs a public parameter prm .
- $\text{KeyGen}_R(\text{prm})$: KeyGen_R is a randomised key-generation algorithm of receivers that on input a public parameter prm , outputs a receiver’s public-key pk_R and a receiver’s secret-key sk_R .
- $\text{KeyGen}_S(\text{prm})$: KeyGen_S is a randomised key-generation algorithm of senders that on input a public parameter prm , outputs a sender’s public-key pk_S and a sender’s secret-key sk_S .
- $\text{SC}(\text{prm}, \text{pk}_R, \text{sk}_S, \mu)$: SC is a randomised signcrypt algorithm that on input a public parameter prm , a receiver’s public-key pk_R , a sender’s secret-key sk_S and a message $\mu \in \mathcal{MSP}$, outputs a ciphertext C .
- $\text{USC}(\text{prm}, \text{pk}_S, \text{sk}_R, C)$: USC is a deterministic unisigncrypt algorithm that on input a public parameter prm , a sender’s public-key pk_S , a receiver’s secret-key sk_R and a ciphertext C , outputs a message μ or an invalid \perp .

It is required that for any $prm \leftarrow \text{Setup}(1^k)$, $(pk_R, sk_R) \leftarrow \text{KeyGen}_R(prm)$ and $(pk_S, sk_S) \leftarrow \text{KeyGen}_S(prm)$, $\mu = \text{USC}(prm, pk_S, sk_R, C)$, where $C \leftarrow \text{SC}(prm, pk_R, sk_S, \mu)$, holds.

The security of signcryption schemes requires both confidentiality and unforgeability. In addition, we consider the insider strong security in the multi-user setting. In the two-user setting, the adversary needs to generate his key-pair at the beginning of security games, and is not allowed to generate other key-pairs. On the other hand, in the multi-user setting, the adversary can generate key-pairs and submit queries with public-keys to the unsigncrypt/signcrypt oracle anytime. Besides, insider has a secret key sk_S (resp. sk_R) in the challenge phase (resp. output phase) of the IND-CCA (resp. sUF-CMA) game. This means that the adversary in the multi-user setting is more powerful than that in the two-user setting.

The notion of confidentiality is MU-IND-iCCA, and notion of unforgeability is MU-sUF-iCMA. These definitions are given as follows.

Definition 4 (MU-IND-iCCA). *MU-IND-iCCA security against a signcryption scheme $\text{SCS} = (\text{Setup}, \text{KeyGen}_R, \text{KeyGen}_S, \text{SC}, \text{USC})$ is defined as follows: Let \mathcal{A} be a PPT adversary against SCS in the following game:*

- **Setup:** The challenger generates $prm \leftarrow \text{Setup}(1^k)$ and $(pk_R, sk_R) \leftarrow \text{KeyGen}_R(prm)$, and sends (prm, pk_R) to \mathcal{A} .
- **Queries 1:** For each query (pk_S, C) which \mathcal{A} submits to the unsigncrypt oracle, the challenger returns $\mu/\perp \leftarrow \text{USC}(prm, pk_S, sk_R, C)$.
- **Challenge:** When \mathcal{A} submits $(\mu_0, \mu_1, pk_S^*, sk_S^*)$, the challenger chooses $b \in \{0, 1\}$ uniformly at random and returns a challenge ciphertext $C^* \leftarrow \text{SC}(prm, pk_R, sk_S^*, \mu_b)$.
- **Queries 2:** For each query (pk_S, C) which \mathcal{A} submits to the unsigncrypt oracle, the challenger does the same processes as the above **Queries 1** phase except that it is required that each query (pk_S, C) meets $(pk_S, C) \neq (pk_S^*, C^*)$.
- **Output:** \mathcal{A} outputs $b' \in \{0, 1\}$, and wins if $b = b'$.

Let $[\mathcal{A} \text{ wins}]$ be an event that \mathcal{A} wins in the above game, and let $\text{Adv}_{\mathcal{A}}^{\text{MU-IND-iCCA}}(k) := |\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}|$ be the advantage of \mathcal{A} . SCS is MU-IND-iCCA secure if $\text{Adv}_{\mathcal{A}}^{\text{MU-IND-iCCA}}(k) \leq \text{negl}(k)$ for any PPT adversary \mathcal{A} .

Definition 5 (MU-sUF-iCMA). *MU-sUF-iCMA security against signcryption scheme $\text{SCS} = (\text{Setup}, \text{KeyGen}_R, \text{KeyGen}_S, \text{SC}, \text{USC})$ is defined as follows: Let \mathcal{A} be a PPT adversary in the following game:*

- **Setup:** The challenger generates $prm \leftarrow \text{Setup}(1^k)$ and $(pk_S, sk_S) \leftarrow \text{KeyGen}_S(prm)$, and sends (prm, pk_S) to \mathcal{A} .
- **Queries:** For each query (pk_R, μ) which \mathcal{A} submits to the signcrypt oracle, the challenger returns $C \leftarrow \text{SC}(prm, pk_R, sk_S, \mu)$. We define that the number of queries that \mathcal{A} submits is at most q .
- **Output:** \mathcal{A} outputs an forgery (pk_R^*, sk_R^*, C^*) , and wins if $(pk_R^*, \mu^*, C^*) \neq (pk_R^i, \mu^i, C^i) \wedge \mu^* = \text{USC}(prm, pk_S, sk_R^*, C^*)$ for $i \in [Q]$ hold.

Let $[\mathcal{A} \text{ wins}]$ be an event that \mathcal{A} wins in the above game, and let $Adv_{\mathcal{A}}^{\text{MU-sUF-iCMA}}(k) := \Pr[\mathcal{A} \text{ wins}]$ be the advantage of \mathcal{A} . SCS is MU-sUF-iCMA secure if $Adv_{\mathcal{A}}^{\text{MU-sUF-iCMA}}(k) \leq \text{negl}(k)$ for any PPT adversary \mathcal{A} .

4 Our Construction Based on Lattice Problems

4.1 Lattice-Based Signcryption

In this section, we propose a lattice-based construction of signcryption. The idea for our construction is as follows: To achieve both of MU-IND-iCCA and MU-sUF-iCMA security, we use a tag-based encryption (TBE) [23], a DS [23], and collision-resistant hash functions [24].

Although our construction is based on sign-then-encrypt methodology, it is shown that the construction, by combining IND-CCA secure PKE (or IND-Tag-CCA secure TBE) and sUF-CMA secure DS in a trivial way of this methodology, cannot achieve MU-sUF-iCMA security while they can meet MU-IND-iCCA security, according to [3,22]. This is because the insider adversary, who has a receiver’s public-key, can unencrypt a ciphertext C by using the signcrypt oracle and obtain a valid pair of messages and signatures which passes verification of the DS. Hence, the adversary can make a forgery in the MU-sUF-iCMA game by encrypting the pair again.

To resolve the problem above, we utilize the following idea in the sign-then-encrypt paradigm: We generate a signature σ not only for a message μ and a receiver’s public-key pk_R , but also for outputs (\bar{c}_0, \bar{c}_1) of tag-based trapdoor (or one-way) functions based on LWE such as $g_A(\mathbf{s}; \mathbf{x}) := \mathbf{s}^T \mathbf{A} + \mathbf{x}^T \bmod q$ for a parameter $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, and an error vector $\mathbf{x} \in \mathbb{Z}^m$. Let $\bar{c}_0 := g_A(\mathbf{s}; \mathbf{x}_0)$ and let $\bar{c}_1 := g_U(\mathbf{s}; \mathbf{x}_1)$. And, it encrypts the signature and the message by computing $\mathbf{c}_0 = \bar{c}_0 + \sigma \bmod q$ and $\mathbf{c}_1 = \bar{c}_1 + \mu \lfloor \frac{q}{2} \rfloor \bmod q$. Then, the adversary needs to generate false \bar{c}_0^* or \bar{c}_1^* to break MU-sUF-iCMA security. However, he cannot generate such a forgery unless he breaks the sUF-CMA security, since \bar{c}_0^* and \bar{c}_1^* are signed.

In addition, our construction uses the lattice-based algorithms, GenTrap, Invert, and SampleD, in Sect. 2.2. We use lattice-based collision-resistant hash functions f in Sect. 2.1.

Our lattice-based construction (i.e., a direct construction based on infeasible lattice problems) LB-SCS = (Setup, KeyGen_R, KeyGen_S, SC, USC) is given as follows:

- $prm \leftarrow \text{Setup}(1^k)$: Set public parameters as follows: a positive integer n ($\gg k$), a prime $q = \text{poly}(n)$, $\bar{m} = O(n \log q)$, $m = \bar{m} + n \log q$, $\alpha^{-1} = O(n \log q)^2 \cdot \omega(\sqrt{\log n})$, $\delta = O(\sqrt{n^2 \log^2 q}) \cdot \omega(\sqrt{\log n})^2$. ℓ is a length of a message, p is a positive integer such that $p = \Omega(q\delta^{-1})$.

$$\bullet \mathbf{G} := \begin{bmatrix} \mathbf{g}^T & 0 \\ & \ddots \\ 0 & \mathbf{g}^T \end{bmatrix} \in \mathbb{Z}_q^{n \times n \log q}, \text{ where } \mathbf{g}^T = [1, 2, 2^2, \dots, 2^{\log q - 1}] \in \mathbb{Z}_q^{1 \times \log q},$$

- The full-rank differences (FRD) encoding [1] $H: \mathbb{Z}_q^n \rightarrow \mathbb{Z}_q^{n \times n}$,
 - $\mathbf{A}_0, \dots, \mathbf{A}_\lambda \stackrel{U}{\leftarrow} \mathbb{Z}_q^{n \times n \log q}$, $\mathbf{B} \stackrel{U}{\leftarrow} \mathbb{Z}_q^{n \times m}$, $\mathbf{U} \stackrel{U}{\leftarrow} \mathbb{Z}_q^{n \times \ell}$, $\mathbf{u}_S \stackrel{U}{\leftarrow} \mathbb{Z}_q^n$.
- Output $prm = (k, n, q, \bar{m}, m, \alpha, \delta, \lambda, \ell, p, \mathbf{G}, H, \mathbf{A}_0, \dots, \mathbf{A}_\lambda, \mathbf{B}, \mathbf{U}, \mathbf{u}_S)$.
- $(pk_R, sk_R) \leftarrow \text{KeyGen}_R(prm)$: Generate a receiver's key-pair as follows:
 1. $\bar{\mathbf{A}}_R \stackrel{U}{\leftarrow} \mathbb{Z}_q^{n \times \bar{m}}$, $\mathbf{T}_R \leftarrow D_{\mathbb{Z}, \log n}^{\bar{m} \times n \log q}$,
 2. $\mathbf{A}_R = [\bar{\mathbf{A}}_R | -\bar{\mathbf{A}}_R \mathbf{T}_R] \in \mathbb{Z}_q^{n \times m}$,
 3. Output $pk_R := \mathbf{A}_R$ and $sk_R := \mathbf{T}_R$.
 - $(pk_S, sk_S) \leftarrow \text{KeyGen}_S(prm)$: Generate a sender's key-pair as follows:
 1. $\bar{\mathbf{A}}_S \stackrel{U}{\leftarrow} \mathbb{Z}_q^{n \times \bar{m}}$, $\mathbf{T}_S \leftarrow D_{\mathbb{Z}, \log n}^{\bar{m} \times n \log q}$,
 2. $\mathbf{A}_S = [\bar{\mathbf{A}}_S | \mathbf{G} - \bar{\mathbf{A}}_S \mathbf{T}_S] \in \mathbb{Z}_q^{n \times m}$,
 3. Output $pk_S := \mathbf{A}_S$ and $sk_S := \mathbf{T}_S$.
 - $C \leftarrow \text{SC}(pk_R, sk_S, \mu)$: To signcrypt $\mu \in \{0, 1\}^\ell$, do the following:
 1. $\mathbf{r}_e, \mathbf{r}_s \leftarrow D_{\mathbb{Z}, \log n}^m$, $\mathbf{t} = f_{\bar{\mathbf{A}}_R}(pk_S) + f_B(\mathbf{r}_e) \in \mathbb{Z}_q^n$,
 2. $\mathbf{A}_{R,t} = [\bar{\mathbf{A}}_R | H(\mathbf{t})\mathbf{G} - \bar{\mathbf{A}}_R \mathbf{T}_R] \in \mathbb{Z}_q^{n \times m}$,
 3. $\mathbf{s} \stackrel{U}{\leftarrow} \mathbb{Z}_q^n$, $\mathbf{x}_0 \leftarrow D_{\mathbb{Z}, \alpha q}^m$, $\mathbf{x}_1 \leftarrow D_{\mathbb{Z}, \alpha q}^\ell$,
 4. $\bar{\mathbf{c}}_0 = \mathbf{s}^T \mathbf{A}_{R,t} + p\mathbf{x}_0^T \in \mathbb{Z}_q^m$, $\bar{\mathbf{c}}_1 = \mathbf{s}^T \mathbf{U} + p\mathbf{x}_1^T \in \mathbb{Z}_q^\ell$,
 5. $\bar{C} = (\bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1, \mathbf{r}_e)$,
 6. Generate a signature on $\mu || pk_R || \bar{C}$.
 - $\mathbf{h} = f_{\bar{\mathbf{A}}_S}(\mu || pk_R || \bar{C}) + f_B(\mathbf{r}_s) \in \mathbb{Z}_q^n$,
 - $\mathbf{A}_{S,h} = [\mathbf{A}_S | \mathbf{A}_0 + \sum_{i=1}^\lambda h_i \cdot \mathbf{A}_i] \in \mathbb{Z}_q^{m+n \log q}$,
 - $\mathbf{e} \leftarrow \text{SampleD}(\mathbf{T}_S, \mathbf{A}_{S,h}, \mathbf{u}_S, \delta)$,
 - $(\mathbf{e}, \mathbf{r}_s) \in \mathbb{Z}^{m+n \log q} \times \mathbb{Z}^m$ is the signature.
 7. $\mathbf{c}_0 = \bar{\mathbf{c}}_0 + \mathbf{r}_s \in \mathbb{Z}_q^m$, $\mathbf{c}_1 = \bar{\mathbf{c}}_1 + p \cdot \mu \lfloor \frac{q}{2} \rfloor \in \mathbb{Z}_q^\ell$,
 8. Output $C = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{r}_e, \mathbf{e})$.
 - $\mu/\perp \leftarrow \text{USC}(pk_S, sk_R, C)$: To unisigncrypt $C = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{r}_e, \mathbf{e})$, do the following:
 1. $\mathbf{t} = f_{\bar{\mathbf{A}}_R}(pk_S) + f_B(\mathbf{r}_e) \in \mathbb{Z}_q^n$, $\mathbf{A}_{R,t} = [\bar{\mathbf{A}}_R | H(\mathbf{t})\mathbf{G} - \bar{\mathbf{A}}_R \mathbf{T}_R] \in \mathbb{Z}_q^{n \times m}$,
 2. $(\mathbf{z}, \mathbf{r}_s) = \text{Invert}(\mathbf{T}_R, \mathbf{A}_{R,t}, \mathbf{c}_0)$,
 3. Compute $\mathbf{E} \in \mathbb{Z}^{m \times \ell}$ s.t. $\mathbf{A}_{R,t} \mathbf{E} = \mathbf{U} \pmod q$ by using the SampleD algorithm,
 4. $\mathbf{v}^T = \mathbf{c}_1^T - (\mathbf{c}_0 - \mathbf{r}_s)^T \mathbf{E} = p(\mathbf{x}_1^T + \mu^T \lfloor \frac{q}{2} \rfloor - \mathbf{x}_0^T \mathbf{E})$,
 5. Recover μ from $\mathbf{v}/p \pmod q$,
 6. $\bar{\mathbf{c}}_0 = \mathbf{c}_0 - \mathbf{r}_s \pmod q$, $\bar{\mathbf{c}}_1 = \mathbf{c}_1 - p\mu \lfloor \frac{q}{2} \rfloor \pmod q$, $\bar{C} = (\bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1, \mathbf{r}_e)$,
 7. $\mathbf{h} = f_{\bar{\mathbf{A}}_S}(\mu || pk_R || \bar{C}) + f_B(\mathbf{r}_s) \in \mathbb{Z}_q^n$, $\mathbf{A}_{S,h} = [\mathbf{A}_S | \mathbf{A}_0 + \sum_{i=1}^\lambda h_i \cdot \mathbf{A}_i] \in \mathbb{Z}_q^{m+n \log q}$,
 8. Output μ if $\mathbf{A}_{S,h} \cdot \mathbf{e} = \mathbf{u}_S \pmod q \wedge \|\mathbf{e}\| \leq \delta \sqrt{m+n \log q}$. Output \perp otherwise.

The following theorems (i.e., Theorems 1 and 2) show security of LB-SCS and their sketch proofs are given below.

Theorem 1. *If $\text{LWE}_{q,\alpha}$ assumption for $\alpha \geq 2\sqrt{n}/q$ holds and collision-resistant hash functions in [10] meet collision-resistance, LB-SCS is MU-IND-iCCA secure.*

Proof. Let \mathcal{A} be a PPT adversary against the signcryption LB-SCS in Sect. 4.1. We consider several MU-IND-iCCA games $Game_i$ for $i \in \{0, 1, 2, 3, 4\}$:

- $Game_0$: This is the normal MU-IND-iCCA game.
- $Game_1$: This game is same as $Game_0$ except that if \mathcal{A} submits an unencrypt query $(pk_S, (c_0, c_1, r_e, e))$ such that $t^* = f_{\bar{A}_R}(pk_S) + f_B(r_e)$, it returns \perp .
- $Game_2$: This game is same as $Game_1$ except that a parameter B of a collision-resistant hash function f_B is replaced with a parameter \mathbf{B} with a trapdoor $T_B \in \mathbb{Z}^{\bar{m} \times n \log q}$ in Proposition 1.
- $Game_3$: This game is same as $Game_2$ except that the challenger generates a value c_0^* of a challenge ciphertext C^* as follows:
 1. $s \xleftarrow{U} \mathbb{Z}_q^n$, $\bar{x}_0 \leftarrow D_{\alpha q}^{\bar{m}}$, $\bar{c}_0^T = s^T \bar{A}_R + p \bar{x}_0^T \pmod q$,
 2. $x'_0 \leftarrow D_{\alpha q}^{n \log q}$, $c_0'^T = -\bar{c}_0^T T_R + p x_0'^T = (s^T (-A_R T_R)) + p(-\bar{x}_0 T_R + x_0')^T \in \mathbb{Z}_q^{n \log q}$.
 3. $\bar{c}_0^{*T} = \hat{c}_0^T || c_0'^T$.
- $Game_4$: This game is same as $Game_3$ except that instead of $\bar{c}_0 = s^T \bar{A}_R + p \bar{x}_0 \pmod q$, $\bar{c}_1 = s^T U + p x_1 \pmod q$, $(\bar{c}_0, \bar{c}_1) \in \mathbb{Z}_q^{\bar{m}} \times \mathbb{Z}_q^{n \times \ell}$ are chosen uniformly at random.

And, we define the following events for $i \in \{0, 1, 2, 3, 4\}$:

- S_i : This is an event that \mathcal{A} wins in $Game_i$.
- V_i : This is an event that in $Game_i$, \mathcal{A} submits an unencrypt query $(pk_S, (c_0, c_1, r_e, e))$ such that $t^* = f_{\bar{A}_R}(pk_S) + f_B(r_e) \in \mathbb{Z}_q^n$.
- CR_i : This is an event that in $Game_i$, \mathcal{A} submits an unencrypt query $(pk_S, (c_0, c_1, r_e, e))$ such that $(pk_S, r_e) \neq (pk_S^*, r_e^*) \wedge t^* = f_{\bar{A}_R}(pk_S) + f_B(r_e) \in \mathbb{Z}_q^n$.
- F_i : This is an event that in $Game_i$, \mathcal{A} submits an unencrypt query $(pk_S^*, (r_e^*, c_0, c_1, e))$ such that $(pk_S, r_e) = (pk_S^*, r_e^*) \wedge \|e\| \leq \delta \sqrt{m + n \log q} \wedge A_{S,h} \cdot e = u_S \pmod q$ for $h \leftarrow f_{\bar{A}_S}(\mu || pk_R || \bar{C}) + f_B(r_s) \in \mathbb{Z}_q^n$.

Then, in the advantage of \mathcal{A} , $|\Pr[S_0] - \Pr[S_1]| \leq \Pr[V_1]$ holds by the difference lemma. $|\Pr[V_1]| \leq \Pr[CR_1] + \Pr[F_1]$ holds by the above definition of these events. Then, we have

$$\begin{aligned} |\Pr[V_1]| &\leq \Pr[CR_1] + \Pr[F_1] \\ &\leq \Pr[CR_1] + \Pr[F_4] + |\Pr[F_3] - \Pr[F_4]| \\ &\quad + |\Pr[F_2] - \Pr[F_3]| + |\Pr[F_1] - \Pr[F_2]| \end{aligned}$$

We show the following Lemmas to prove that $Adv_{\mathcal{A}}^{\text{MU-IND-iCCA}}(k)$ is negligible.

Lemma 1. $\Pr[CR_1] \leq \text{negl}(k)$.

By using \mathcal{A} , we construct a PPT algorithm \mathcal{B}_0 breaking the collision-resistance of $f_{\bar{A}_R} + f_B$ in the following way:

- **Setup:** Take $(\mathbf{B}, \bar{\mathbf{A}}_R) \in \mathbb{Z}_q^{n \times (m+\bar{m})}$ as input. Generate \mathbf{A}_R as follows: Sample $\mathbf{T}_R \in \mathbb{Z}_q^{\bar{m} \times n \log q}$ from $D_{\sqrt{\log n}}^{\bar{m} \times n \log q}$, compute $\mathbf{A}_R \leftarrow [\bar{\mathbf{A}}_R \mid -\bar{\mathbf{A}}_R \mathbf{T}_R]$. Choose $\mathbf{U} \in \mathbb{Z}_q^{n \times \ell}$ uniformly at random. Add (\mathbf{U}, \mathbf{B}) to a public parameter, let $pk_R := \mathbf{A}_R$ and let $sk_R = \mathbf{T}_R$. Run $\mathcal{A}(prm, pk_R)$.
- **Queries 1:** When \mathcal{A} submits a query $(pk_S, (\mathbf{c}_0, \mathbf{c}_1, \mathbf{e}, \mathbf{r}_e))$ to the unencrypt oracle, do the same processes as $Game_0$.
- **Challenge:** When \mathcal{A} submits a challenge query $(pk_S^*, sk_S^*, \mu_0, \mu_1)$, compute $\mathbf{t}^* = f_B(\mathbf{r}_e^*) + f_{\bar{\mathbf{A}}_R}(pk_S^*) \in \mathbb{Z}_q^n$. Choose $b \in \{0, 1\}$ and compute a challenge ciphertext C^* on μ_b in the same way as the SC algorithm.
- **Queries 2:** When \mathcal{A} submits a query $(pk_S, (\mathbf{c}_0, \mathbf{c}_1, \mathbf{e}, \mathbf{r}_e))$ to the unencrypt oracle, do the following: Check whether the event CR_1 happens. If so, halt and output a pair $((\mathbf{r}_e^*, pk_S^*), (\mathbf{r}_e, pk_S))$. Otherwise, return μ/\perp according to the USC algorithm.
- **Output:** When \mathcal{A} outputs the guessing bit $b' \in \{0, 1\}$, output a random bit in $\{0, 1\}$.

\mathcal{B}_0 simulates the view of \mathcal{A} completely. When \mathcal{A} submits a query such that $\mathbf{t}^* = f_{\bar{\mathbf{A}}_R}(pk_S) + f_B(\mathbf{r}_e) \in \mathbb{Z}_q^n$ and $(pk_s, \mathbf{r}_e) \neq (pk_s, \mathbf{r}_e)$, the output of \mathcal{B}_0 is a collision pair against the collision-resistant function $f_{\bar{\mathbf{A}}_R} + f_B$ clearly. \square

Lemma 2. $|\Pr[S_1] - \Pr[S_2]| \leq \text{negl}(k)$, $|\Pr[F_1] - \Pr[F_2]| \leq \text{negl}(k)$, $|\Pr[S_2] - \Pr[S_3]| \leq \text{negl}(k)$ and $|\Pr[F_2] - \Pr[F_3]| \leq \text{negl}(k)$.

$|\Pr[S_1] - \Pr[S_2]|$ and $|\Pr[F_1] - \Pr[F_2]|$ are negligible by Proposition 1. $|\Pr[S_2] - \Pr[S_3]|$ and $|\Pr[F_2] - \Pr[F_3]|$ are negligible by using Corollary 3.10 of [30] and Theorem 3.1 of [27]. \square

Lemma 3. $|\Pr[S_3] - \Pr[S_4]| \leq \text{negl}(k)$, $|\Pr[F_3] - \Pr[F_4]| \leq \text{negl}(k)$, $|\Pr[S_4] - \frac{1}{2}| \leq \text{negl}(k)$, and $\Pr[F_4] \leq \text{negl}(k)$.

$|\Pr[S_4] - \frac{1}{2}|$ and $\Pr[F_4]$ are negligible since μ and \mathbf{r}_S are statistically hidden in $Game_4$.

We prove that $|\Pr[S_3] - \Pr[S_4]| \leq \text{negl}(k)$ by constructing a PPT algorithm \mathcal{B}_1 solving LWE problem. We construct this algorithm in the following way:

- **Setup:** Take a LWE challenge $(\bar{\mathbf{A}}_R, \mathbf{U}, \bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1) \in \mathbb{Z}_q^{n \times (\bar{m}+\ell)} \times \mathbb{Z}_q^{\bar{m}+\ell}$ as input. Choose $\mathbf{t}^* \in \mathbb{Z}_q^n$ uniformly at random and generate $(\mathbf{B}, \mathbf{T}_B) \leftarrow \text{TrapGen}(1^k)$. Sample $\mathbf{T}_R \in \mathbb{Z}^{\bar{m} \times n \log q}$ from $D_\delta^{\bar{m} \times n \log q}$. Let $\mathbf{A}_R := [\bar{\mathbf{A}}_R \mid -H(\mathbf{t}^*)\mathbf{G} - \bar{\mathbf{A}}_R \mathbf{T}_R]$. Add $(\bar{\mathbf{A}}, \mathbf{U}, \mathbf{B})$ to a public parameter. Let $pk_R := \mathbf{A}_R$ and $sk_R := (\mathbf{T}_R, \mathbf{T}_B)$. Run $\mathcal{A}(prm, pk_R)$.
- **Queries 1:** When \mathcal{A} submits an unencrypt query $(pk_S, (\mathbf{c}_0, \mathbf{c}_1, \mathbf{e}, \mathbf{r}_e))$, respond in the same way as $Game_3$ by sk_R and the Invert algorithm of Proposition 1.
- **Challenge:** When \mathcal{A} submits a challenge query $(pk_S^*, sk_S^*, \mu_0, \mu_1)$, do the following
 1. $b \xleftarrow{U} \{0, 1\}$, $\mathbf{r}_e^* \leftarrow \text{SampleD}(\mathbf{T}_B, \mathbf{B}, (\mathbf{t}^* - f_{\bar{\mathbf{A}}_R}(pk_S^*)) \bmod q, \log n)$,
 2. $\mathbf{c}_0'^T = \bar{\mathbf{c}}_0^T \mathbf{T}_R + \mathbf{x}_0'^T$, $\bar{\mathbf{c}}_0'^T = p(\bar{\mathbf{c}}_0^T || \mathbf{c}_0'^T) \in \mathbb{Z}_q^m$,
 3. $\bar{\mathbf{c}}_1^* = p\bar{\mathbf{c}}_1 \in \mathbb{Z}_q^\ell$.

4. Compute $(\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{e}^*, \mathbf{r}_e^*)$ by (pk_S^*, sk_S^*) in the same way as the SC algorithm,
 5. Return $C^* := (\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{e}^*, \mathbf{r}_e^*)$.
- Queries 2: Do the same process as the Queries 1 phase. However, \mathcal{A} submits a query (pk_S, C) such that $(pk_S, C) \neq (pk_S^*, C^*)$.
 - Output: When \mathcal{A} outputs the guessing bit $b' \in \{0, 1\}$, output 1 if $b' = b$. Output 0 otherwise.

We analyze the above algorithm \mathcal{B}_1 . When \mathcal{A} submits queries such that $\mathbf{t} \neq \mathbf{t}^*$, it is possible to return μ/\perp following the USC algorithm because $H(\mathbf{t}) - H(\mathbf{t}^*) = H(\mathbf{t} - \mathbf{t}^*) \in \mathbb{Z}_q^{n \times n}$ is invertible by the property of the FRD encoding [1] and we can use the Invert algorithm. In the Queries 1 phase, the probability that \mathcal{A} submits queries such that $\mathbf{t} = \mathbf{t}^*$ is negligible since \mathbf{t}^* is statistically hidden in the Setup phase. \mathcal{B}_1 simulates the view of \mathcal{A} completely. By using the output of \mathcal{A} , \mathcal{B}_1 can determine whether $(\bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1)$ is the sample of LWE or not. Hence, it is possible to solve the LWE problem.

We can also prove that $|\Pr[F_3] - \Pr[F_4]| \leq \text{negl}(k)$ by constructing a PPT algorithm \mathcal{B}_2 solving LWE. This algorithm is same as \mathcal{B}_1 except that \mathcal{B}_2 checks whether \mathcal{A} submitted an unsigncrypt query such that $\|e\| \leq \delta\sqrt{m+n}\log q$ and $\mathbf{A}_{S,h} \cdot \mathbf{e} = \mathbf{u}_S \bmod q$, and outputs 0 if so. It outputs 1 otherwise. \square

From the above lemmas, $\text{Adv}_{\mathcal{A}}^{\text{MU-IND-iCCA}}(k)$ is negligible, and hence the proof is completed. \square

Theorem 2. *If $\text{SIS}_{q,\beta}$ assumption for $\beta = O((n \log n)^{5/2}) \cdot \omega(\log n^{3/2})$ holds and hash functions meet collision-resistance, LB-SCS is MU-sUF-iCMA secure.*

Proof. Let \mathcal{A} be a PPT adversary against SCS in the MU-sUF-iCMA game, let $M := \mu \|pk_R\|_{c_0}$, and let $x^{(i)}$ for $i \in [Q]$ be a value x generated for i th oracle access.

The forger \mathcal{A} can be classified into several types as follows.

Type-1. \mathcal{A} generates a forgery by finding a collision of $f_{\bar{\mathbf{A}}_S} + f_{\mathbf{B}}$.

Type-2. \mathcal{A} generates a forgery without finding a collision of $f_{\bar{\mathbf{A}}_S} + f_{\mathbf{B}}$.

(a) \mathcal{A} generates a forgery without any queried message.

(b) \mathcal{A} generates a new forgery by using a queried message.

When \mathcal{A} is a Type-1 adversary, we construct a PPT adversary \mathcal{S} breaking collision-resistance of $f_{\bar{\mathbf{A}}_S} + f_{\mathbf{B}}$ and show that $\text{Adv}_{\mathcal{A}}^{\text{MU-sUF-iCMA}}(k) \leq \text{Adv}_{\mathcal{S}}^{\text{CR}}(k)$. \mathcal{S}_{CR} is constructed as follows:

- Setup: Take matrices $\bar{\mathbf{A}}_S \in \mathbb{Z}_q^{n \times \bar{m}}$ and $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$ as input. In the same way as the Setup and KeyGen $_S$ algorithm, generate prm and $(pk_S, sk_S) := ((\mathbf{A}_S, \mathbf{u}_S), \mathbf{T}_S)$ by using $\bar{\mathbf{A}}$ and \mathbf{B} . Run $\mathcal{A}(prm, pk_S)$.
- Queries: When \mathcal{A} submits (pk_R, μ) as a query, compute c_0 by using pk_R . Choose \mathbf{r}_s at random and compute $\mathbf{h} = f_{\bar{\mathbf{A}}_S}(\mu \|pk_R\|_{c_0}) + f_{\mathbf{B}}(\mathbf{r}_s)$. Compute a ciphertext $C := (\mathbf{c}_0, \mathbf{c}_1, \mathbf{r}_e, \mathbf{e})$ on μ following the SC algorithm. Then, send C to \mathcal{A} .

- **Output:** When \mathcal{A} outputs a forgery (pk_R^*, sk_R^*, C^*) , compute μ^* following the USC algorithm and output $(M^*, \mathbf{r}_s^*), (M^{(i)}, \mathbf{r}_s^{(i)})$ such that $f_{\bar{A}_S}(M^*) + f_B(\mathbf{r}_s^*) = f_{\bar{A}_S}(M^{(i)}) + f_B(\mathbf{r}_s^{(i)})$ with $(M^*, \mathbf{r}_s^*) \neq (M^{(i)}, \mathbf{r}_s^{(i)})$.

Then, \mathcal{S}_{CR} simulates the environment of \mathcal{A} completely. The forgery (pk_R^*, sk_R^*, C^*) is generated by finding a collision of $f_{\bar{A}_S} + f_B$ and \mathcal{S} 's output is a collision of CH clearly. Hence, $Adv_{\mathcal{A}}^{\text{MU-sUF-iCMA}}(k) \leq Adv_{\mathcal{S}_{CR}}^{\text{CR}}(k)$ holds.

Next, we consider Type-2 adversary and show a reduction from $\text{SIS}_{q,\beta}$ for $\beta = O(\lambda(n \log q)^{3/2}) \cdot \omega(\sqrt{\log n})^3$ to MU-sUF-iCMA of SCS. We make a modification that the adversary cannot distinguish. We replace a parameter/trapdoor-pair $(\mathbf{B}, \mathbf{T}_B)$ with a parameter \mathbf{B} of a collision-resistant hash function.

First, We consider Type-2-(a) adversary, that is, adversary outputs a forgery on a never queried message. By using this adversary, we construct \mathcal{S}_1 solving $\text{SIS}_{q,\beta}$. This algorithm is given $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{A}'] \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{u} \in \mathbb{Z}_q^n$, and outputs $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{z} = \mathbf{u} \pmod q$ and $\|\mathbf{z}\| \leq \beta - 1$. A PPT algorithm \mathcal{S}_1 solving $\text{SIS}_{q,\beta}$ is as follows.

- **Setup:** Take $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{A}'] \in \mathbb{Z}_q^{n \times (\bar{m} + n \log q)}$ as input. To generate prm and (pk_S, sk_S) , do the following:
 - Choose messages $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \dots, \mathbf{h}^{(Q)} \in \{0, 1\}^\lambda$ uniformly at random. Compute the set \mathcal{P} of all strings P such that the length $|P| \leq \lambda$ and there is no $\mathbf{h}^{(i)}$ including P as a prefix.
 - Compute $(\mathbf{A}_S, \mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_\lambda, \mathbf{u})$ in the following way: For $i \in \{0, 1, \dots, \lambda\}$, choose $\mathbf{T}_{S,i} \leftarrow D_{\log n}^{\bar{m} \times n \log q}$ and compute $\mathbf{A}_i = \mathbf{H}_i \mathbf{G} - \bar{\mathbf{A}} \mathbf{T}_{S,i}$. \mathbf{H}_i is

$$\mathbf{H}_i = \begin{cases} H(\mathbf{0}) = \mathbf{0} & (i > t) \\ (-1)^{P_i} \cdot H(\mathbf{u}^{(i)}) & (i \in [t]) \\ -\sum_{j \in [t]} P_j \cdot \mathbf{H}_j & (i = 0) \end{cases}$$

- For $\mathbf{h}^{(i)}$, generate a signature on $\mu^{(i)}$ as follows:
 1. $\mathbf{A}_{S,\mathbf{h}^{(i)}} \leftarrow [\mathbf{A} \mid \mathbf{A}_0 + \sum_{j \in [\lambda]} h_j^{(i)} \mathbf{A}_j] = [\bar{\mathbf{A}}_S \mid \mathbf{A}' \mid \mathbf{H}\mathbf{G} - \bar{\mathbf{A}}(\mathbf{T}_{S,0} + \sum_{j \in [\lambda]} h_j^{(i)} \mathbf{T}_{S,j})]$,
 2. Let $\mathbf{T}_S^{(i)} = \mathbf{T}_{S,0} + \sum_{j \in [\lambda]} h_j^{(i)} \mathbf{T}_{S,j}$ be a trapdoor for the matrix $\mathbf{A}_{S,\mathbf{h}^{(i)}}$,
 3. Compute a signature $\mathbf{e}^{(i)} \leftarrow \text{SampleD}(\mathbf{T}_S^{(i)}, \mathbf{A}_{S,\mathbf{h}^{(i)}}, \mathbf{u}_S, \delta)$.
- **Queries:** For each query $(\mu^{(i)}, pk_R^{(i)})$ to the signcrypt oracle, do the following:
 1. Compute $\bar{C}^{(i)}$ following the SC algorithm,
 2. $\mathbf{r}_s^{(i)} \leftarrow \text{SampleD}(\mathbf{T}_B, \mathbf{B}, \mathbf{h}^{(i)} - f_{\bar{A}_S}(M^{(i)}) \pmod q, \log n)$,
 3. Compute $(\mathbf{c}_0^{(i)}, \mathbf{c}_1^{(i)})$ by using a signature $(\mathbf{r}_s^{(i)}, \mathbf{e}^{(i)})$,
 4. Return $C^{(i)} := (\mathbf{c}_0^{(i)}, \mathbf{c}_1^{(i)}, \mathbf{r}_e^{(i)}, \mathbf{e}^{(i)})$.
- **Output:** When \mathcal{A} outputs a forgery $(pk_R^*, sk_R^*, C^* = (\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{r}_e^*, \mathbf{e}^*))$, check that it is possible to unsigncrypt μ^* by the USC algorithm and compute \mathbf{e} and $\mathbf{h}^* \leftarrow f_{\bar{A}_S}(M^*) + f_B(\mathbf{r}_s^*) \pmod q$.

We show that \mathcal{S}_1 can solve SIS problem by using a pair $(\mathbf{h}^*, \mathbf{e}^*)$. Since a prefix of \mathbf{h}^* is P , the following holds: Since $\mathbf{A}_{S, \mathbf{h}^*} = [\bar{\mathbf{A}}_S \mid \mathbf{A}' \mid -\bar{\mathbf{A}}_S(\mathbf{T}_{S,0} + \sum_{j \in [\lambda]} h_j^{(i)} \mathbf{T}_{S,j})] = [\bar{\mathbf{A}}_S \mid \mathbf{A}' \mid -\bar{\mathbf{A}}_S \mathbf{T}_{S, \mathbf{h}^*}]$, we have

$$\underbrace{[\bar{\mathbf{A}}_S \mid \mathbf{A}']}_{\mathbf{A}_S} \underbrace{\begin{bmatrix} \mathbf{I}_m & -\mathbf{T}_S^* \\ & \mathbf{I}_{n \log q} \end{bmatrix}}_z \mathbf{y}^* = \mathbf{u}_S \pmod{q}.$$

$\|\mathbf{y}^*\| \leq \delta\sqrt{m} = O(\sqrt{\lambda n \log q}) \cdot \omega(\sqrt{\log n})^2$ holds obviously, and $\|\mathbf{T}_S^*\| \leq \sqrt{\lambda + 1} \cdot O(\sqrt{m} + \sqrt{n \log q}) \cdot \omega(\sqrt{\log n})$ holds from Lemma 2.9 of [23]. Hence, $\|\mathbf{z}\| = O(\lambda(n \log q)^{3/2}) \cdot \omega(\sqrt{\log n})^3 \leq \beta - 1$.

Next, we consider a Type-2-(b) adversary. That is, this is that \mathcal{A} generates a new forgery against a queried \mathbf{h}^* . We construct \mathcal{S}_2 solving SIS in the same way as the \mathcal{S}_1 algorithm except for the process of the Setup and Output phase. This algorithm is given $\mathcal{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{u} \in \mathbb{Z}_q^n$, and outputs $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{z} = \mathbf{0} \pmod{q}$ and $\|\mathbf{z}\| \leq \beta$. We describe the process of \mathcal{S}_2 in Setup phase as follows:

- Choose $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \dots, \mathbf{h}^{(Q)} \in \{0, 1\}^\lambda$ uniformly at random, and choose $\mathbf{h}^{(i)}$ from these at random and let $\mathbf{h} := \mathbf{h}^{(i)}$.
- Generate $(\mathbf{A}_S, \mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_\lambda, \mathbf{u}_S)$ as follows: Generate \mathbf{A}_i in the same way as the \mathcal{S}_1 and $p \leftarrow \mathbf{h}$. Choose $\mathbf{y} \leftarrow D_{\mathbb{Z}, \delta}$ and let $\mathbf{u}_S := \mathbf{A}_\mathbf{h} \mathbf{y} \pmod{q}$.
- For $i \in [Q]$, generate a signature $\mathbf{e}^{(i)}$ on $\mathbf{h}^{(i)}$ in the same way as \mathcal{S}_1 except for \mathbf{h} . Let \mathbf{y} be a signature on \mathbf{h} .

We describe processes of \mathcal{S}_2 in Output phase in the following way: When \mathcal{A} outputs (pk_R^*, sk_R^*, C^*) , compute $(\mathbf{h}^*, \mathbf{e}^*)$ by using the USC algorithm. Since $\mathbf{h}^* = \mathbf{h}$, the following holds: Since $\mathbf{A}_{S, \mathbf{h}^*} = [\bar{\mathbf{A}}_S \mid \mathbf{A}' \mid -\bar{\mathbf{A}}_S(\mathbf{T}_{S,0} + \sum_{j \in [\lambda]} h_j^{(i)} \mathbf{T}_{S,j})] = [\bar{\mathbf{A}}_S \mid \mathbf{A}' \mid -\bar{\mathbf{A}}_S \mathbf{T}_{S, \mathbf{h}^*}]$, we have

$$\underbrace{[\bar{\mathbf{A}}_S \mid \mathbf{A}']}_{\mathbf{A}_S} \underbrace{\begin{bmatrix} \mathbf{I}_{\bar{m}} & -\mathbf{T}_S^* \\ & \mathbf{I}_{n \log q} \end{bmatrix}}_z (\mathbf{y}^* - \mathbf{y}) = \mathbf{0} \pmod{q}.$$

$\|\mathbf{y}\|, \|\mathbf{y}^*\| \leq \delta\sqrt{m} = O(\sqrt{\lambda n \log q}) \cdot \omega(\sqrt{\log n})^2$ holds clearly, and $\|\mathbf{T}_S^*\| = O(\sqrt{\lambda n \log q}) \cdot \omega(\sqrt{\log n})$. Hence, $\|\mathbf{z}\| = O(\lambda(n \log q)^{3/2}) \cdot \omega(\sqrt{\log n})^3 = O((n \log q)^{5/2}) \cdot \omega(\sqrt{n})^3$.

From the discussion about Type-2-(a) and Type-2-(b) adversary, we completed the reduction from $\text{SIS}_{q, \beta}$.

From the above, the proof was completed. \square

4.2 Lattice-Based Hybrid Signcryption

In this section, we propose a lattice-based hybrid signcryption obtained by combining LB-SCS in Sect. 4.1 and a DEM. In LB-SCS, the ciphertext size for a

message is $|\mu| \log q$ for the bit-length of a message $|\mu|$ and a modulus q . By combining LB-SCS with a DEM, the ciphertext size for a message μ is reduced to $|\mu|$.

Let $\text{DEM} = (\text{DEM.Enc}, \text{DEM.Dec})$ be an IND-OT secure DEM meeting one-to-one property². Our hybrid signcryption $\text{HSC} = (\text{Setup}, \text{KeyGen}_R, \text{KeyGen}_S, \text{SC}, \text{USC})$ is as follows. The Setup , KeyGen_R , and KeyGen_S algorithms of HSC are the same as those of LB-SCS.

- $C \leftarrow \text{SC}(pk_R, sk_S, \mu)$: To signcrypt $\mu \in \{0, 1\}^{|\mu|}$, do the following:
 1. $K \xleftarrow{U} \{0, 1\}^\ell$, where ℓ is the bit-length of a DEM's symmetric key,
 2. $\mathbf{r}_e, \mathbf{r}_s \leftarrow D_{\mathbb{Z}, \log n}^m$, $\mathbf{t} = f_{\bar{A}_R}(pk_S) + f_B(\mathbf{r}_e) \in \mathbb{Z}_q^n$,
 3. $\mathbf{A}_{R,t} = [\bar{A}_R \mid H(\mathbf{t})\mathbf{G} - \bar{A}\mathbf{T}_R] \in \mathbb{Z}_q^{n \times m}$,
 4. $\mathbf{s} \xleftarrow{U} \mathbb{Z}_q^n$, $\mathbf{x}_0 \leftarrow D_{\mathbb{Z}, \alpha q}^m$, $\mathbf{x}_1 \leftarrow D_{\mathbb{Z}, \alpha q}^\ell$,
 5. $\bar{\mathbf{c}}_0 = \mathbf{s}^T \mathbf{A}_{R,t} + p\mathbf{x}_0^T \in \mathbb{Z}_q^m$, $\bar{\mathbf{c}}_1 = \mathbf{s}^T \mathbf{U} + p\mathbf{x}_1^T \in \mathbb{Z}_q^\ell$,
 6. $\bar{C} = (\bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1, \mathbf{r}_e)$,
 7. Generate a signature on $\mu \parallel K \parallel pk_R \parallel \bar{C}$.
 - $\mathbf{h} = f_{\bar{A}_S}(\mu \parallel K \parallel pk_R \parallel \bar{C}) + f_B(\mathbf{r}_s) \in \mathbb{Z}_q^n$,
 - $\mathbf{A}_{S,h} = [\mathbf{A}_S \mid \mathbf{A}_0 + \sum_{i=1}^\lambda h_i \cdot \mathbf{A}_i] \in \mathbb{Z}_q^{m+n \log q}$,
 - $\mathbf{e} \leftarrow \text{SampleD}(\mathbf{T}_S, \mathbf{A}_{S,h}, \mathbf{u}_S, \delta)$,
 - $(\mathbf{e}, \mathbf{r}_s) \in \mathbb{Z}^{m+n \log q} \times \mathbb{Z}^m$ is the signature.
 8. $\mathbf{c}_0 = \bar{\mathbf{c}}_0 + \mathbf{r}_s \in \mathbb{Z}_q^m$, $\mathbf{c}_1 = \bar{\mathbf{c}}_1 + p \cdot K \lfloor \frac{q}{2} \rfloor \in \mathbb{Z}_q^\ell$,
 9. $\mathbf{c}_2 = \text{DEM.Enc}(K, \mu)$,
 10. Output $C = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{r}_e, \mathbf{e})$.
- $\mu/\perp \leftarrow \text{USC}(pk_S, sk_R, C)$: To unsigncrypt $C = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{r}_e, \mathbf{e})$, do the following:
 1. $\mathbf{t} = f_{\bar{A}_R}(pk_S) + f_B(\mathbf{r}_e) \in \mathbb{Z}_q^n$, $\mathbf{A}_{R,t} = [\bar{A}_R \mid H(\mathbf{t})\mathbf{G} - \bar{A}\mathbf{T}_R] \in \mathbb{Z}_q^{n \times m}$,
 2. $(\mathbf{z}, \mathbf{r}_s) = \text{Invert}(\mathbf{T}_R, \mathbf{A}_{R,t}, \mathbf{c}_0)$,
 3. Compute $\mathbf{E} \in \mathbb{Z}^{m \times \ell}$ s.t. $\mathbf{A}_{R,t} \mathbf{E} = \mathbf{U} \pmod q$ by using the SampleD algorithm,
 4. $\mathbf{v}^T = \mathbf{c}_1^T - (\mathbf{c}_0 - \mathbf{r}_s)^T \mathbf{E} = p(\mathbf{x}_1^T + K^T \lfloor \frac{q}{2} \rfloor - \mathbf{x}_0^T \mathbf{E})$,
 5. Recover K from $\mathbf{v}/p \pmod q$,
 6. $\mu = \text{DEM.Dec}(K, \mathbf{c}_2)$,
 7. $\bar{\mathbf{c}}_0 = \mathbf{c}_0 - \mathbf{r}_s \pmod q$, $\bar{\mathbf{c}}_1 = \mathbf{c}_1 - p \cdot K \lfloor \frac{q}{2} \rfloor \pmod q$, $\bar{C} = (\bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1, \mathbf{r}_e)$,
 8. $\mathbf{h} = f_{\bar{A}_S}(\mu \parallel K \parallel pk_R \parallel \bar{C}) + f_B(\mathbf{r}_s) \in \mathbb{Z}_q^n$, $\mathbf{A}_{S,h} = [\mathbf{A}_S \mid \mathbf{A}_0 + \sum_{i=1}^\lambda h_i \cdot \mathbf{A}_i] \in \mathbb{Z}_q^{m+n \log q}$,
 9. Output μ if $\mathbf{A}_{S,h} \cdot \mathbf{e} = \mathbf{u}_S \pmod q \wedge \|\mathbf{e}\| \leq \delta \sqrt{m+n \log q}$. Output \perp otherwise.

The above construction HSC is shown to be secure by the following theorems.

Theorem 3. *If $\text{LWE}_{q,\alpha}$ assumption for $\alpha \geq 2\sqrt{n}/q$ holds and DEM is IND-OT secure, HSC is MU-IND-iCCA secure.*

² We say that a DEM meets one-to-one if for a message μ and a symmetric-key K , there is only one ciphertext c such that $\mu = \text{DEM.Dec}(K, c)$.

Proof. Let \mathcal{A} be a PPT adversary in the MU-IND-iCCA game. We define the following games.

- *Game₀*: This is the ordinary MU-IND-iCCA game.
- *Game₁*: This is the same game as *Game₀* except that if \mathcal{A} submits a decryption query $(\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{r}_e, \mathbf{e})$ such that $(\mathbf{c}_0, \mathbf{c}_1, \mathbf{r}_e, \mathbf{e}) = (\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{r}_e^*, \mathbf{e}^*)$ and $\mathbf{c}_2 \neq \mathbf{c}_2^*$, the oracle does not use the symmetric-key, but uses the key K^* generated by the SC algorithm.
- *Game₂*: This is the same game as *Game₁* except that the symmetric-key K is chosen uniformly at random.

Let S_i be an event that \mathcal{A} wins in *Game_i* for $i \in \{0, 1, 2\}$.

Game₁ is identical to *Game₀* unless it is possible to generate key-pairs $(pk_R, sk_R), (pk_S, sk_S)$ such that, for some symmetric key K generated in the SC algorithm, K cannot be obtained following the USC algorithm. Therefore, we have $|\Pr[S_0] - \Pr[S_1]| \leq \text{negl}(k)$ by completeness of HSC.

We can prove $|\Pr[S_1] - \Pr[S_2]| \leq 2Adv^{LWE}(k)$ in the same way as in Theorem 1. The reason is as follows: A symmetric key K is signcrypted in the same way as LB-SCS in Sect. 4.1 and $|\Pr[S_1] - \Pr[S_2]| = 2Adv_{LB-SCS}^{\text{MU-IND-iCCA}}(k)$ holds. Hence, $|\Pr[S_1] - \Pr[S_2]| \leq 2Adv^{LWE}(k)$ holds.

We show $|\Pr[S_2] - \frac{1}{2}| \leq Adv_{DEM}^{\text{IND-OT}}(k)$. We can simulate Setup, Queries 1, and Queries 2 phases, following the algorithms of HSC. In the Challenge phase, we encrypt the ciphertext \mathbf{c}_2^* by sending messages to the challenger in the IND-OT game. In the Output phase, the guessing bit $b' \in \{0, 1\}$ in the IND-OT game is the same as the output of \mathcal{A} in the MU-IND-iCCA game.

From the above discussion, $Adv_{HSC}^{\text{MU-IND-iCCA}}(k) \leq 2Adv^{LWE}(k) + Adv_{DEM}^{\text{IND-OT}}(k)$. Therefore, the proof was completed. \square

Theorem 4. *If SIS_{q,\beta}} assumption for $\beta = O((n \log n)^{5/2}) \cdot \omega(\log n^{3/2})$ holds and DEM is one-to-one, HSC is MU-sUF-iCMA secure.*

Proof. It is possible to prove this theorem by the same way as Theorem 2. The reason is as follows:

- If an adversary tries to make a forgery on a queried message μ , he has to make a new symmetric-key K which was not used in the Queries phase, because DEM is one-to-one. However, K has to be signcrypted in the same way as LB-SCS. Hence, he needs to break SIS problem.
- Suppose that an adversary tries to make a forgery on a message μ which was never queried. However, in HSC, the message has to be signcrypted in the same way as LB-SCS. Hence, he has to break the SIS problem in this case as well.

Therefore, by using the same proof technique in Theorem 2, we finally get $Adv_{HSC}^{\text{MU-sUF-iCMA}}(k) \leq Adv^{SIS}(k)$. \square

5 Comparison of Lattice-Based Signcryption

We compare our scheme with other existing ones in terms of key-sizes (i.e., sizes of public-keys and secret-keys), and ciphertext-size in order to evaluate efficiency among the constructions, where our scheme means the construction in Sect. 4.2 (i.e., more efficient one).

To the best of our knowledge, our constructions are the first direct constructions of signcryption based on lattice problems without random oracles. Hence, there is no other direct construction to compare efficiency with ours. However, since there are generic constructions of signcryption [11, 25] satisfying the strongest security (i.e., both of MU-IND-iCCA and MU-sUF-iCMA) without random oracles, we can obtain lattice-based constructions of signcryption by applying suitable lattice-based primitives to the generic constructions. Specifically, we consider the following applications of lattice-based primitives.

\mathcal{SC}_{TK} [11]: we apply IND-Tag-CCA secure Tag-based KEM [10, 23], sUF-CMA secure DS [10, 23], IND-CCA secure DEM.

\mathcal{SC}_{KEM} [11]: we apply IND-CCA secure KEM [7, 23], sUF-CMA secure DS [10, 23], IND-OT secure DEM, sUF-OT secure MAC.

\mathcal{SC}_{CHK} [25]: we apply IND-sID-CPA secure Identity-based Encryption [1], UF-CMA secure DS [8], sUF-OT secure OTS [21].

In the description above, MAC is a message authentication code, and OTS is a one-time signature. IND-Tag-CCA means indistinguishability against adaptive tag chosen ciphertext attack, sUF-OT means strong unforgeability against one-time attack, and IND-sID-CPA means indistinguishability against selective ID chosen plaintext attack. Then, to fairly compare efficiency of lattice-based signcryption, we take into account the following:

1. In \mathcal{SC}_{TK} , \mathcal{SC}_{KEM} , and our scheme, we assume that the paradigm of authenticated encryption in [5] is used to obtain IND-CCA secure DEM (or IND-OT secure DEM). Namely, IND-CCA secure DEM is obtained from IND-CPA secure symmetric-key encryptions (SKE) and sUF-CMA secure MAC. These SKE and MAC can be constructed from the AES meeting the 128-bit security and IND-CPA security. The key sizes are set to be at least 512 bits, since it is necessary to have resistance against quantum computing by taking into account the power of the Grover's algorithm.
2. In \mathcal{SC}_{TK} , IND-Tag-CCA secure tag-based KEM is required. However, there is no lattice-based construction meeting this security. We construct this tag-based KEM by combining tag-based KEM achieving weaker security (see Appendix B) and a chameleon hash function [10] in a generic way.
3. In \mathcal{SC}_{KEM} , we construct IND-CCA secure KEM by the BK-transformation [7]. This reason is that even if we consider realizing CCA-secure KEM based on the lattice problems [26], the resulting signcryption will be less efficient than \mathcal{SC}_{TK} and \mathcal{SC}_{CHK} obviously.
4. In \mathcal{SC}_{CHK} , a lattice-based one-time signature [21] is required. The construction in [21] requires the ideal-lattice assumptions, which is believed to be

stronger than the standard SIS and LWE assumption (i.e, Definitions 1 and 2). However, in an extended version of [21], a framework of lattice-based OTS is proposed. By using this framework, we can obtain sUF-OT secure OTS based on the SIS assumption and we apply this construction here.

Table 1. Comparison of sizes of public/secret keys and ciphertexts [bit]: A positive integer n is a security parameter, q is a prime, a positive integer $m = O(n \log q)$ is a dimension of a lattice, $d(\ll q)$ is a value of an element sampled from a Gaussian distribution in \mathbb{Z} , $|MAC|$ is the bit-length of MAC tags, K is the bit-length of symmetric keys of DEM, $|vk|$ is the bit-length of an OTS’s verification-key, and $|\mu|$ is the bit-length of a message.

Construction	Receiver’s key size		Sender’s key size		Ciphertext size
	Public key	Secret key	Public key	Secret key	
\mathcal{SC}_{TK}	$3nm \log q$	$nm \log q \log d$	$3nm \log q$	$nm \log q \log d$	$(m + K) \log q + 3m \log d + \mu $
\mathcal{SC}_{KEM}	$2nm \log q$				$(2m + K + n) \log q + 2m \log d + \mu + 2 MAC $
\mathcal{SC}_{CHK}	$nm \log q$		$nm \log q$		$(3m + n) \log q + vk + \mu \log q$
Our scheme					$(m + K) \log q + 2m \log d + \mu $

Table 2. Parameter setting.

Parameters	Size
n	256
q	4093
$m = 3n \log q$	9215
$d = 2n \log q \cdot \log n$	49148
$ vk \approx n^2 \log^2 n$	42.0×10^5
K	512
$ MAC $	128

Table 3. Comparison of ciphertext-size.

Constructions	Ciphertext-size (bit-length)
\mathcal{SC}_{TK}	5.5×10^5
\mathcal{SC}_{KEM}	5.2×10^5
\mathcal{SC}_{CHK}	45.3×10^5
Our scheme	4.0×10^5

Table 1 shows comparison in sizes of public/secret-keys and ciphertexts. Although it can be seen that \mathcal{SC}_{CHK} and our scheme are most efficient in terms of receiver’s public-key size, it is difficult to conclude which construction is best in terms of ciphertext size, only from Table 1. Therefore, we evaluate the constructions by applying concrete parameters. Table 2 shows evaluation results of

parameters which we estimated by referring to [19]. Table 3 shows the result obtained by applying the parameters in Table 2 to the constructions of signcryption. From Table 3, we can observe that ciphertext-size of our scheme is shortest.

From the above discussion, the public-key size and ciphertext size in our scheme are shorter than any other schemes, and there is no disadvantage for ours in other parameters compared to other ones. Furthermore, even if other setting parameters in [2] are applied, we can similarly observe the advantage of our scheme compared to other schemes as well.

Appendix A: Data Encapsulation Mechanism (DEM)

A DEM consists of a two-tuple of polynomial-time algorithms $DEM = (Enc, Dec)$ as follows: Let \mathcal{MSP} be a message-space, and let \mathcal{KSP} be a key-space.

- $Enc(K, \mu)$: Enc is a randomised encryption algorithm that on input a symmetric-key $K \in \mathcal{KSP}$ and a message $\mu \in \mathcal{MSP}$, outputs a ciphertext C .
- $Dec(K, C)$: Dec is a deterministic decryption algorithm that on input a secret-key K and a ciphertext C , outputs a message μ or invalid \perp .

IND-OT security against $DEM = (Enc, Dec)$ is defined as follows: Let \mathcal{A} be a PPT adversary in the following game.

- **Setup**: The challenger generates a symmetric-key $K \xleftarrow{U} \mathcal{KSP}$.
- **Challenge**: When \mathcal{A} submits (μ_0, μ_1) , the challenger chooses $b \in \{0, 1\}$ uniformly at random and returns $C^* \leftarrow Enc(K, \mu_b)$.
- **Output**: \mathcal{A} outputs $b' \in \{0, 1\}$, and wins if $b = b'$.

Let $Adv_{\mathcal{A}}^{IND-OT}(k) := |\Pr[b = b'] - \frac{1}{2}|$ be the advantage of \mathcal{A} . DEM is IND-OT secure if $Adv_{\mathcal{A}}^{IND-OT}(k) \leq \text{negl}(k)$ for any PPT adversary \mathcal{A} .

Appendix B: Lattice-Based Tag-Based Encryption and Tag-Based KEM

IND-sTag-CCA secure TBE based on LWE can be easily constructed from lattice-based tag-based trapdoor functions [23], where IND-sTag-CCA means indistinguishability against selective tag chosen ciphertext attack (cf. [17]). Hence, we obtain the following lattice-based construction of $TBE = (Setup, Kg, Enc, Dec)$:

- $prm \leftarrow Setup(1^n)$: Take a security parameter n as input and then set public parameters prm as follows: a prime $q = \text{poly}(n)$, integers $\bar{m} = O(n \log q)$, $m = \bar{m} + n \log q$, $\alpha^{-1} = O(n \log q)^2 \cdot \omega(\sqrt{\log \bar{m}})$, a matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times n \log q}$ is as the definition in Sect. 2.2.

Output $prm = (n, q, \bar{m}, m, \alpha, \mathbf{G})$.

- $(pk, sk) \leftarrow \text{Kg}(prm)$: To generate a public-key pk and a secret-key sk , do the following:
 1. $\bar{\mathbf{A}} \xleftarrow{U} \mathbb{Z}_q^{n \times \bar{m}}, \mathbf{T} \leftarrow D_{\log n}^{\bar{m} \times n \log q}, \mathbf{U} \xleftarrow{U} \mathbb{Z}_q^{n \times \ell}, \mathbf{A} = [\bar{\mathbf{A}} | -\bar{\mathbf{A}}\mathbf{T}] \in \mathbb{Z}_q^{n \times m}$,
 2. Output $pk = (\mathbf{A}, \mathbf{U}), sk = \mathbf{T}$.
- $C \leftarrow \text{Enc}(pk, tag, \mu)$: To encrypt a message $\mu \in \{0, 1\}^\ell$, do the following:
 1. $\mathbf{s} \xleftarrow{U} \mathbb{Z}_q^n, \mathbf{x}_0 \leftarrow D_{\alpha q}^m, \mathbf{x}_1 \leftarrow D_{\alpha q}^\ell, \mathbf{A}_{tag} = [\bar{\mathbf{A}} | H(tag)\mathbf{G} - \bar{\mathbf{A}}\mathbf{T}] \in \mathbb{Z}_q^{n \times m}$,
 2. $\mathbf{c}_0 = \mathbf{s}^T \mathbf{A}_{tag} + \mathbf{x}_0^T \in \mathbb{Z}_q^m, \mathbf{c}_1 = \mathbf{s}^T \mathbf{U} + \mathbf{x}_1^T + \mu \cdot \lfloor q/2 \rfloor \in \mathbb{Z}_q^\ell$,
 3. Output a ciphertext $C = (\mathbf{c}_0, \mathbf{c}_1)$.
- $\mu \leftarrow \text{Dec}(sk, tag, C)$: To decrypt $C = (\mathbf{c}_0, \mathbf{c}_1)$, do the following:
 1. $\mathbf{A}_{tag} = [\bar{\mathbf{A}} | H(tag)\mathbf{G} - \bar{\mathbf{A}}\mathbf{T}], (\mathbf{s}, \mathbf{x}_0) = \text{Invert}(\mathbf{T}, \mathbf{A}_{tag}, \mathbf{c}_0)$,
 2. $\mathbf{d} = \mathbf{c}_1 - \mathbf{s}^T \mathbf{U} \in \mathbb{Z}_q^\ell$, let $\mathbf{d} = (d_1, \dots, d_\ell)$, and for each $i \in [\ell], k_i = 0$ if d_i is closer to 0 than to $\lfloor \frac{q}{2} \rfloor$, otherwise let $k_i = 1$.
 3. Output a message $\mu = (k_1, \dots, k_\ell) \in \{0, 1\}^\ell$.

Note that the above construction is based on LWE, and we can obtain the lattice-based (IND-sTag-CCA secure) TB-KEM from the TBE above by replacing a message with a random key.

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_28
2. Albrecht, M.R., Cid, C., Faugère, J., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. Des. Codes Crypt. **74**(2), 325–354 (2015)
3. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 83–107. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_6
4. Baek, J., Steinfeld, R., Zheng, Y.: Formal proofs for the security of signcryption. J. Cryptol. **20**(2), 203–235 (2007)
5. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. J. Cryptol. **21**(4), 469–491 (2008)
6. Böhl, F., Hofheinz, D., Jäger, T., Koch, J., Striecks, C.: Confined guessing: new signatures from standard assumptions. J. Cryptol. **28**(1), 176–208 (2015)
7. Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM J. Comput. **36**(5), 1301–1328 (2007)
8. Boyen, X.: Lattice mixing and vanishing trapdoors: a framework for fully secure short signatures and more. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 499–517. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_29
9. Boyen, X., Li, Q.: Towards tightly secure lattice short signature and Id-based encryption. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 404–434. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_14
10. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. J. Cryptol. **25**(4), 601–639 (2012)

11. Chiba, D., Matsuda, T., Schuldt, J.C.N., Matsuura, K.: Efficient generic constructions of signcryption with insider security in the multi-user setting. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 220–237. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21554-4_13
12. 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
13. Ducas, L., Micciancio, D.: Improved short lattice signatures in the standard model. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 335–352. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_19
14. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC 2008, pp. 197–206. ACM (2008)
15. Goldreich, O., Goldwasser, S., Halevi, S.: Collision-free hashing from lattice problems. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 3, no. 42 (1996)
16. Kawachi, A., Tanaka, K., Xagawa, K.: Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 372–389. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_23
17. Kiltz, E.: Chosen-ciphertext security from tag-based encryption. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 581–600. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_30
18. Libert, B., Quisquater, J.-J.: Efficient signcryption with key privacy from gap Diffie-Hellman groups. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 187–200. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24632-9_14
19. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19074-2_21
20. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43
21. Lyubashevsky, V., Micciancio, D.: Asymptotically efficient lattice-based digital signatures. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 37–54. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_3
22. Matsuda, T., Matsuura, K., Schuldt, J.C.N.: Efficient constructions of signcryption schemes and signcryption composability. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT 2009. LNCS, vol. 5922, pp. 321–342. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10628-6_22
23. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_41
24. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.* **37**(1), 267–302 (2007)
25. Nakano, R., Shikata, J.: Constructions of signcryption in the multi-user setting from identity-based encryption. In: Stam, M. (ed.) IMACC 2013. LNCS, vol. 8308, pp. 324–343. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45239-0_19

26. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: STOC 2009, pp. 333–342. ACM (2009)
27. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 80–97. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_5
28. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_31
29. Peikert, C., Waters, B.: Lossy trapdoor functions and their applications. In: STOC 2008, pp. 187–196. ACM (2008)
30. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. *J. ACM* **56**(6), 34:1–34:40 (2009)
31. Rückert, M.: Strongly unforgeable signatures and hierarchical identity-based signatures from lattices without random oracles. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 182–200. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12929-2_14
32. Tan, C.H.: Signcryption scheme in multi-user setting without random oracles. In: Matsuura, K., Fujisaki, E. (eds.) IWSEC 2008. LNCS, vol. 5312, pp. 64–82. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89598-5_5
33. Yamada, S.: Adaptively secure identity-based encryption from lattices with asymptotically shorter public parameters. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 32–62. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_2
34. Yan, J., Wang, L., Wang, L., Yang, Y., Yao, W.: Efficient lattice-based signcryption in standard model. *Math. Prob. Eng.* **2013** (2013)
35. Zhang, J., Chen, Y., Zhang, Z.: Programmable hash functions from lattices: short signatures and IBEs with small key sizes. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 303–332. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_11
36. Zheng, Y.: Digital signcryption or how to achieve $\text{cost}(\text{signature} \ \& \ \text{encryption}) \ll \text{cost}(\text{signature}) + \text{cost}(\text{encryption})$. In: Kaliski B.S. (eds) *Advances in Cryptology – CRYPTO 1997*, CRYPTO 1997. LNCS, vol. 1294, pp. 165–179. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0052234>

Multivariate Cryptography



Rank Analysis of Cubic Multivariate Cryptosystems

John Baena¹, Daniel Cabarcas^{1(✉)}, Daniel E. Escudero², Karan Khathuria³,
and Javier Verbel¹

¹ Universidad Nacional de Colombia sede Medellín, Medellín, Colombia
{jbbaena,dcabarc,javerbelh}@unal.edu.co

² Aarhus University, Aarhus, Denmark
escudero@cs.au.dk

³ University of Zurich, Zurich, Switzerland
karan.khathuria@math.uzh.ch

Abstract. In this work we analyze the security of cubic cryptographic constructions with respect to rank weakness. We detail how to extend the big field idea from quadratic to cubic, and show that the same rank defect occurs. We extend the min-rank problem and propose an algorithm to solve it in this setting. We show that for fixed small rank, the complexity is even lower than for the quadratic case. However, the rank of a cubic polynomial in n variables can be larger than n , and in this case the algorithm is very inefficient. We show that the rank of the differential is not necessarily smaller, rendering this line of attack useless if the rank is large enough. Similarly, the algebraic attack is exponential in the rank, thus useless for high rank.

Keywords: Multivariate cryptography · Cubic polynomials
Tensor rank · Min-rank

1 Introduction

The min-rank problem (MR) is, given k $m \times n$ matrices and a target rank r , to determine whether there exists a linear combination of the matrices of rank less or equal to r . Although NP-complete in its general setting, there are efficient algorithms to solve it for certain parameters. Indeed, Kipnis and Shamir modeled an attack on the HFE system as an MR problem and were able to break it. Since then, other multivariate public key schemes (MPK) have been subject to similar attacks. Rank defects also lead to other weakness such as a fixed degree of regularity in the algebraic attack on HFE [6].

The importance of the rank itself, and the prevalence of MR as an attack technique in MPK suggest a more central role as the underlying problem that

D. E. Escudero—Work done whilst at Universidad Nacional de Colombia.

supports security. For example, we can think of HFE as a way to construct low rank quadratic polynomials. Their low rank allows inversion, but it is insecure because the same low rank is preserved as a linear combination of the public key which can be efficiently solved through the Kipnis-Shamir modeling (KS) of MR.

Although the MR problem is stated for two-dimensional matrices, it can be naturally extended to d -dimensional matrices. It is particularly interesting to analyze it for three-dimensional matrices, since rank problems become much harder there. For example, simply determining the rank of a matrix is difficult for three-dimensional matrices, and it is not even known the maximum possible rank a matrix may have (see e.g. [15]).

Three-dimensional matrices lead to cubic polynomials. They are less common than quadratic polynomials in MPKs for two reasons. First, they are larger thus less efficient than quadratics. But more important, if f is cubic, its differential $Df_{\mathbf{a}}(\mathbf{x}) := f(\mathbf{x} + \mathbf{a}) - f(\mathbf{x}) - f(\mathbf{a})$ is a quadratic map that preserves some of the properties of f . Thus, it is possible to extend rank analysis techniques from quadratics to cubics targeting the differential, c.f. [26]. Yet one important question remains open: Is this a general property of any cubic map that dooms any such construction? In this paper we address this question, by taking a general perspective not focused on a particular construction.

1.1 Our Contribution

In order to close the knowledge gap, we gather the appropriate literature to frame the discussion of the rank of cubic polynomials. We use the language of tensors that allows for very natural extensions of key concepts from two to d -dimensional matrices.

We extend the MR problem to three-dimensional matrices and we propose two ways to solve it, which naturally extend the KS modeling. Interestingly, if the rank is small, the complexity is even lower than for the quadratic case. However, the rank of a cubic polynomial in n variables can be larger than n , and in this case the attack is very inefficient.

We also discuss the relevance of two other typical lines of attack for MPK in the context of cubic low rank polynomials, namely the algebraic and differential attacks. We show that the rank of the differential is not necessarily much smaller than the rank of the cubic polynomial, rendering this line of attack inefficient if the rank is large enough. Similarly, the algebraic attack is exponential in the rank, thus useless for high rank.

Although our approach is general, we provide a detailed example. We show how to efficiently construct cubic polynomials over a finite field from a weight three polynomial over a field extension, extending the so called big field idea. And then, we show that the rank is preserved by this construction in the sense that, a low rank core polynomial leads to a set of cubic polynomials with a low rank linear combination.

1.2 Related Work

In [25, 26], Moody et al. do a rank analysis of the cubic ABC scheme [7]. They expose a subspace differential invariant extending the ideas used in the quadratic case [24]. They show that the MR attack used in [24] can be adapted to this cubic case.

Their work avoids discussing the rank of cubic polynomials by focusing on the differentials. This is rewarding in the ABC case because of the band structure of the scheme. There are linear combinations of the public polynomials with a band structure (they show it for the second differential) whose rank is bounded (possibly by a factor of s^2). The rank of some of their slices (or the second differential evaluated at some vectors as they show) drops by a square root factor to $2s$. This allows an attack on cubic ABC even more efficient than on its quadratic counterpart.

For a good reason, they approach the MR problem by guessing kernel vectors instead of using the Kipnis-Shamir or minors modeling (see Sect. 2.4 for a discussion of these techniques). The subspace differential invariant allows a tight analysis of the efficiency of this approach.

2 Preliminaries

2.1 Notation

Given a natural number n , the set $\{1, \dots, n\}$ is denoted by $[n]$. Let \mathbb{F} be a finite field of order q which, unless explicitly stated, has characteristic different from 2 or 3. Vectors are denoted by bold letters, e.g. \mathbf{u}, \mathbf{v} , and they are treated as column vectors by default unless stated otherwise. The vector \mathbf{e}_i denotes the i -th canonical vector, i.e. the vector whose only non-zero entry is the i -th one, which is equal to 1. The i -th entry of a vector \mathbf{u} is denoted by $\mathbf{u}[i]$, but sometimes we also use the non-bold version of the corresponding letter with subscript i : u_i . The space of all $n \times m$ matrices is denoted by $\mathbb{F}^{n \times m}$. The entry of a matrix A indexed by (i, j) is denoted by $A[i, j]$. We use the notation $A[i, \cdot]$ to refer to the i -th row of a matrix A (as a row vector), and $A[\cdot, j]$ to refer to the j -th column of A (as a column vector). A three dimensional matrix of dimensions $n \times m \times \ell$ is an array of elements in \mathbb{F} indexed by tuples (i, j, k) , where $1 \leq i \leq n$, $1 \leq j \leq m$ and $1 \leq k \leq \ell$. The vector space of these three-dimensional matrices is denoted by $\mathbb{F}^{n \times m \times \ell}$, and the entry indexed by (i, j, k) in a matrix $A \in \mathbb{F}^{n \times m \times \ell}$ will be denoted by $A[i, j, k]$. We denote by $A[i, \cdot, \cdot]$ the two-dimensional matrix whose entry (j, k) is given by $A[i, j, k]$, and similarly for $A[\cdot, j, \cdot]$ and $A[\cdot, \cdot, k]$. For $\mathbf{u} \in \mathbb{F}^n$ and $\mathbf{v} \in \mathbb{F}^m$, $\mathbf{u} \otimes \mathbf{v}$ denotes the Kronecker product which we usually see as the matrix $\mathbf{u}\mathbf{v}^\top$.

2.2 Rank and Trilinear Forms

Let n, m, l be positive integers and let U, V and W be the vector spaces $\mathbb{F}^n, \mathbb{F}^m$ and \mathbb{F}^l , respectively. The rank of a matrix $A \in \mathbb{F}^{n \times m}$ can be defined as the minimum number of summands r required to write A as

$$A = \sum_{i=1}^r \mathbf{u}_i \otimes \mathbf{v}_i,$$

where $\mathbf{u}_i \in U$ and $\mathbf{v}_i \in V$ for all $i = 1, \dots, r$. This definition of rank is more flexible than other definitions as it is independent of the number of dimensions so it can be extended to three-dimensional matrices as follows.

Definition 1. Let $A \in \mathbb{F}^{n \times m \times \ell}$ be a three-dimensional matrix, we define the rank of A as the minimum number of summands r required to write A as

$$A = \sum_{i=1}^r \mathbf{u}_i \otimes \mathbf{v}_i \otimes \mathbf{w}_i,$$

where $\mathbf{u}_i \in U$, $\mathbf{v}_i \in V$ and $\mathbf{w}_i \in W$ for all $i = 1, \dots, r$. We denote this number by $\text{Rank}(A)$.

Let $A \in \mathbb{F}^{n \times m \times \ell}$ be a three-dimensional matrix. Then clearly, $\text{Rank}(A) = 0$ if and only if A is zero (empty sum). For an arbitrary $A \in \mathbb{F}^{n \times n \times n}$, the maximal value that $\text{Rank}(A)$ can attain is unknown. To our knowledge, the best known upper bound for the maximal value of $\text{Rank}(A)$ is $\lceil (3/4)n^2 \rceil$ (see [17, Theorem 7]).

A bilinear map $B : U \times U \rightarrow \mathbb{F}$ is a map that is linear in each argument, so it can be written as

$$B(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A \mathbf{y} \tag{1}$$

where $A \in \mathbb{F}^{n \times n}$ is the matrix such that $A[i, j] = B(\mathbf{e}_i, \mathbf{e}_j)$.

A bilinear map B is symmetric if for all $\mathbf{a}, \mathbf{b} \in U$ it holds that $B(\mathbf{a}, \mathbf{b}) = B(\mathbf{b}, \mathbf{a})$, which is equivalent to A being symmetric.

Given a bilinear map B we can obtain a quadratic homogeneous polynomial $f(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$ by defining $f(\mathbf{x}) := B(\mathbf{x}, \mathbf{x})$. Different bilinear maps can yield the same quadratic polynomial. Yet, symmetric bilinear maps are in bijection with the set of quadratic homogeneous polynomials. The symmetric bilinear map from a quadratic homogeneous polynomial f can be computed as $B(\mathbf{x}, \mathbf{y}) := \frac{1}{2}(f(\mathbf{x} + \mathbf{y}) - f(\mathbf{x}) - f(\mathbf{y}))$.

Similarly, a trilinear map $T : U \times U \times U \rightarrow \mathbb{F}$ is a map that is linear in each argument. It can be written as

$$T(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{i,j,k \in [n]} x_i y_j z_k \cdot \alpha_{i,j,k}$$

where $\alpha_{i,j,k} := T(\mathbf{e}_i, \mathbf{e}_j, \mathbf{e}_k)$. Let $A \in \mathbb{F}^{n \times n \times n}$ be such that $A[i, j, k] = \alpha_{i,j,k}$. We say that T is symmetric if for all $\mathbf{a}, \mathbf{b}, \mathbf{c} \in U$, it is invariant under any permutation of the indices, i.e.

$$T(\mathbf{a}, \mathbf{b}, \mathbf{c}) = T(\mathbf{a}, \mathbf{c}, \mathbf{b}) = T(\mathbf{b}, \mathbf{a}, \mathbf{c}) = T(\mathbf{c}, \mathbf{a}, \mathbf{b}) = T(\mathbf{b}, \mathbf{c}, \mathbf{a}) = T(\mathbf{c}, \mathbf{b}, \mathbf{a}),$$

or equivalently, the three-dimensional matrix A is symmetric. Given a trilinear form T (symmetric or not) we can obtain the homogeneous cubic polynomial

$f(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$ defined as $f(\mathbf{x}) := T(\mathbf{x}, \mathbf{x}, \mathbf{x})$, and given a homogeneous polynomial f of degree 3 we can obtain the corresponding symmetric trilinear form as

$$T(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \frac{1}{3!}(f(\mathbf{x} + \mathbf{y} + \mathbf{z}) - f(\mathbf{y} + \mathbf{z}) - f(\mathbf{x} + \mathbf{z}) - f(\mathbf{x} + \mathbf{y}) + f(\mathbf{x}) + f(\mathbf{y}) + f(\mathbf{z})). \quad (2)$$

For a cubic homogeneous polynomial $f \in \mathbb{F}[\mathbf{x}]$, we define its rank, denoted by $\text{Rank}(f)$, as the rank of the corresponding three-dimensional symmetric matrix.

2.3 Big Field Idea

Let n be a positive integer. Let $g(y) = y^n + a_{n-1}y^{n-1} + \dots + a_1y + a_0$ be an irreducible polynomial of degree n over \mathbb{F} . Consider the degree n field extension $\mathbb{K} = \mathbb{F}[y]/(g(y))$. Notice that \mathbb{K} can be seen as a vector space over \mathbb{F} of dimension n , so $\mathbb{K} \cong \mathbb{F}^n$ through the usual vector space isomorphism $\phi : \mathbb{K} \rightarrow \mathbb{F}^n$ given by

$$\phi(u_1 + u_2y + \dots + u_ny^{n-1}) = (u_1, u_2, \dots, u_n).$$

Let Δ be the matrix whose i -th row is given by the Frobenius powers $((y^0)^{q^{i-1}}, (y^1)^{q^{i-1}}, \dots, (y^{n-1})^{q^{i-1}})$.

The matrix Δ , whose transpose is known as a Moore matrix, is invertible because $\{y^0, y^1, \dots, y^{n-1}\}$ is a basis of \mathbb{K} over \mathbb{F} ([21], p. 109).

For $\beta \in \mathbb{K}$ let $\text{Fr}(\beta)$ denote the vector $(\beta, \beta^q, \dots, \beta^{q^{n-1}}) \in \mathbb{K}^n$. If $\alpha \in \mathbb{K}$, then it is easy to see that $\text{Fr}(\alpha) = \Delta \cdot \phi(\alpha)$.

We refer to a polynomial in $\mathbb{K}[X]$ of the form

$$\mathcal{F}(X) = \sum_{0 \leq i_1 \leq \dots \leq i_d \leq n-1} \alpha_{i_1, \dots, i_d} X^{q^{i_1} + \dots + q^{i_d}}$$

where $\alpha_{i_1, \dots, i_d} \in \mathbb{K}$ as a homogeneous weight d polynomial. Notice that a homogeneous weight 0 polynomial is simply a constant polynomial, i.e. an element of \mathbb{K} . A weight d polynomial $\mathcal{F} \in \mathbb{K}[X]$ is a polynomial that can be written as $\mathcal{F} = \mathcal{F}_0 + \dots + \mathcal{F}_d$ where each $\mathcal{F}_j \in \mathbb{K}[X]$ is a homogeneous weight j polynomial.

The main property of this type of polynomials is that if $\mathcal{F} \in \mathbb{K}[X]$ is homogeneous of weight d then the map $F = \phi \circ \mathcal{F} \circ \phi^{-1} : \mathbb{F}^n \rightarrow \mathbb{F}^n$ can be represented as evaluation of n homogeneous multivariate polynomials in $\mathbb{F}[x_1, \dots, x_n]$ of degree d . We state this formally in the following theorem.

Theorem 1. *Let $\mathcal{F} \in \mathbb{K}[X]$ be a homogeneous weight d polynomial. There exist homogeneous degree d polynomials $f_1, \dots, f_n \in \mathbb{F}[x_1, \dots, x_n]$ such that for all $\mathbf{a} \in \mathbb{F}^n$ it holds that $F(\mathbf{a}) = (f_1(\mathbf{a}), \dots, f_n(\mathbf{a}))^\top$ where F is the composition $\phi \circ \mathcal{F} \circ \phi^{-1}$.*

Proof (sketch). Since for any $j \in \{0, 1, \dots, n-1\}$, $X \mapsto X^{q^j}$ is an \mathbb{F} -linear map over \mathbb{K} , and $X^{q^{i_1} + \dots + q^{i_d}} = X^{q^{i_1}} \dots X^{q^{i_d}}$, then it is easy to see that each component of F is a degree d multivariate polynomial over \mathbb{F} . A more detailed proof can be found in [8] as Theorem 6.2.1. □

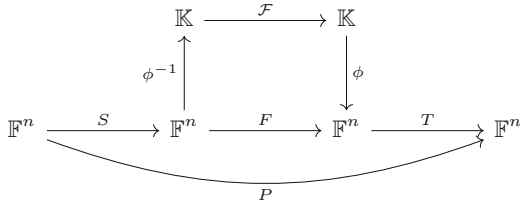


Fig. 1. Big Field Idea

The previous property has been used extensively in order to generate sequences of multivariate quadratic polynomials (f_1, \dots, f_n) that can be inverted with the help of some secret information. Usually, some weight 2 polynomial $\mathcal{F} \in \mathbb{K}[X]$ is chosen, along with two invertible matrices $S, T \in \mathbb{F}^{n \times n}$. The previous theorem states that the composition $F = \phi \circ \mathcal{F} \circ \phi^{-1}$ is given by n multivariate quadratic polynomials in $\mathbb{F}[x_1, \dots, x_n]$ and therefore the composition $P = T \circ F \circ S = T \circ \phi \circ \mathcal{F} \circ \phi^{-1} \circ S$ is also given by n multivariate quadratic polynomials in $\mathbb{F}[x_1, \dots, x_n]$. Usually, \mathcal{F} is referred as the core or central polynomial. If we ensure that \mathcal{F} is a univariate polynomial that is easy to invert, then we can invert P if we know S, T and \mathcal{F} . This construction can be observed in Fig. 1.

This type of “trapdoor” polynomials yield public key encryption schemes where the public key is the polynomials themselves, the secret key is the trapdoor information that allows the inversion of the polynomials, encryption is just evaluation and decryption is inversion. This concept is known as the Big Field Idea, and some examples of schemes that follow this paradigm are MI [23], HFE [27], and variants of HFE [5, 28, 29].

An important remark is that the polynomials representing the map F can be efficiently computable from the coefficients of the polynomial \mathcal{F} . The construction for $d = 2$ can be found in [8, Sect. 6.3]. We will show the construction for $d = 3$ in Sect. 4.

2.4 Two-Dimensional MinRank Attack

Buss et al. [4] introduced the min-rank problem (MR) in the context of linear algebra and proved its NP-completeness.

Definition 2 (MinRank Problem). *Given positive integers m, n, r, k , and matrices $M_0, \dots, M_k \in \mathcal{M}_{m \times n}(\mathbb{F})$, determine whether there exist $\lambda_1, \dots, \lambda_k \in \mathbb{F}$ such that the rank of $\sum_{i=1}^k \lambda_i M_i - M_0$ is less or equal to r .*

In the context of cryptography MR first appeared as part of an attack against the HFE cryptosystem by Kipnis and Shamir [18]. The HFE cryptosystem, proposed in 1996 by Jacques Patarin [27], is based on the big field idea presented in Sect. 2.3, with a low rank central polynomial $\mathcal{F} \in \mathbb{K}[X]$. Kipnis and Shamir showed that an attack on HFE can be reduced to an instance of MR with a small

rank r . In particular, if $M_1, \dots, M_n \in \mathbb{F}^{n \times n}$ are the symmetric matrices representing public polynomials, then there exists a linear combination $\sum_{i=1}^n \lambda_i M_i$ having rank at most the rank of \mathcal{F} . Moreover, these coefficients can be used to construct an equivalent secret key. For more details on the MinRank attack on HFE we refer the reader to [18]. We discuss below some of the most common approaches to solve the min-rank problem.

The Kipnis-Shamir Modeling. Let $A = \sum_{i=1}^k t_i M_i - M_0$ be the matrix with entries in the polynomial ring $\mathbb{F}[t_1, \dots, t_k]$. Then it is easy to see that the matrix A has rank at most r if and only if the dimension of its right kernel is at least $n - r$. Hence we construct $(n - r)$ linearly independent vectors in the right kernel of M by solving the following system of equations in $\mathbb{F}[t_1, \dots, t_k, v_{1,1}, \dots, v_{r,n-r}]$:

$$A \cdot \begin{pmatrix} I_{n-r} \\ V \end{pmatrix} = 0_{n \times (n-r)}, \tag{3}$$

where V is the matrix given by $V[i, j] = v_{i,j}$ for $i = 1, \dots, r$ and $j = 1, \dots, n - r$. This relation produces a system of $n(n - r)$ bi-homogeneous polynomials of bi-degree (1,1) in $k + r(n - r)$ variables. Clearly, if $(t_1, \dots, t_k, v_{1,1}, \dots, v_{r,n-r})$ is a solution of the system, then the evaluation of the matrix A at the point (t_1, \dots, t_k) has rank at most r .

Guessing Kernel Vectors. As with any system of equations, it is possible to guess some variables in (3) and solve for the others. Because of the structure of this system, it is particularly appealing to guess kernel vectors (i.e. the $v_{i,j}$ variables) and solve the resulting linear system in the t_i variables, as proposed in [13] (in fact, since the linear system is very overdetermined, it is enough to guess k/n kernel vectors). The complexity of such attack is dominated by the guessing part and depends on the probability of a correct guess. A tight bound on this probability can be significantly improved by understanding the structure of the solution space, e.g. by exploiting the interlinked kernel structure [33] or by using the subspace differential invariant structure [24].

The Minors Modeling. In [12], Faugère et al. introduced the minors method approach to solve the min-rank problem and in [3] they improved the MinRank attack on HFE using this modeling. Let $M = \sum_{i=1}^k t_i M_i$ be the matrix with entries in the polynomial ring $\mathbb{F}[t_1, \dots, t_k]$. Let I be the ideal in $\mathbb{F}[t_1, \dots, t_k]$ generated by all the $(r + 1) \times (r + 1)$ minors of M . Let $V(I) \subseteq \mathbb{F}^k$ be the zero locus of I . If $(\lambda_1, \dots, \lambda_k) \in V(I) \cap \mathbb{F}^k$, then all the $(r + 1) \times (r + 1)$ minors of the matrix M evaluated at $(\lambda_1, \dots, \lambda_k)$ are zero. As a result the rank of the matrix M evaluated at $(\lambda_1, \dots, \lambda_k)$ is at most r . Each $(r + 1)$ -minor is a homogeneous polynomial in $\mathbb{F}[t_1, \dots, t_k]$ of degree $r + 1$, and the number of $(r + 1)$ -minors in M is $\binom{n}{r+1}^2$.

3 Rank Analysis of Cubic Polynomials

Despite the disadvantages in terms of efficiency of considering cubic polynomials, one possible advantage would be avoiding the MinRank attack on the quadratic case. This might be expected since the MinRank attack relies on the fact that the degree is 2. For instance, this allows us to represent the polynomials as $\mathbf{x}^\top \mathbf{A} \mathbf{x}$, which is crucial as the attack performs matrix operations and properties of such. Therefore, a natural question is whether or not the MinRank attack applies in a cubic setting. Let us start by defining the MinRank problem in this context.

Definition 3 (Cubic MinRank Problem). *Given positive integers l, m, n, r, k , and three-dimensional matrices $M_0, \dots, M_k \in \mathbb{F}^{n \times m \times l}$, determine whether there exist $\lambda_1, \dots, \lambda_k \in \mathbb{F}$ such that the rank of $\sum_{i=1}^k \lambda_i M_i - M_0$ is less or equal to r .*

In this section we show that if there is a low-rank linear combination of the cubic polynomials of the public key then the resulting instance of the MinRank problem can be solved with an extension of the Kipnis-Shamir modeling. This is by itself a weakness on the scheme as it allows an adversary to distinguish between a public key and a random polynomial system of equivalent size. Thereafter, we discuss other consequences of the low-rank for the differentials and for the direct algebraic attack.

3.1 Solving the Three-Dimensional Min-Rank Problem

The following characterization of rank for cubic matrices leads to a generalization of the Kipnis-Shamir modeling for the min-rank problem (for a proof, see e.g. [20]).

Theorem 2. *Given a three-dimensional matrix $A \in \mathbb{F}^{n \times m \times \ell}$, the rank of A is the minimal number r of rank one matrices $S_1, \dots, S_r \in \mathbb{F}^{m \times \ell}$, such that, for all slices $A[i, \cdot, \cdot]$ of A , $A[i, \cdot, \cdot] \in \text{span}(S_1, \dots, S_r)$.*

Let $M_0, \dots, M_k \in \mathbb{F}^{n \times n \times n}$. Then, $A = \sum_{i=1}^k \lambda_i M_i - M_0$ is of rank r , if and only if, there exist rank one matrices $S_1, \dots, S_r \in \mathbb{F}^{n \times n}$, such that, for $i = 1, \dots, n$, $A[i, \cdot, \cdot] \in \text{span}(S_1, \dots, S_r)$. Since each S_i matrix is of rank one, we can write it as $S_i = \mathbf{u}_i \mathbf{v}_i^T$ for some vectors $\mathbf{u}_i, \mathbf{v}_i \in \mathbb{F}^n$. Considering the entries of the \mathbf{u}_i 's, \mathbf{v}_i 's, and the linear combination coefficients as variables yields a cubic system of n^3 equations in $r(2n) + rn + k = 3rn + k$ variables

$$\sum_{j=1}^r \alpha_{ij} \mathbf{u}_j \mathbf{v}_j^T = A[i, \cdot, \cdot], \text{ for } i = 1, \dots, n. \tag{4}$$

If $r \ll n$ we can do much better. In that case, for most such rank r matrices A , the first r slices $A[1, \cdot, \cdot], \dots, A[r, \cdot, \cdot]$ are linearly independent. In this case, $\text{span}(S_1, \dots, S_r) = \text{span}(A[1, \cdot, \cdot], \dots, A[r, \cdot, \cdot])$. Then, for $i = r + 1, \dots, n$, $A[i, \cdot, \cdot] \in \text{span}(A[1, \cdot, \cdot], \dots, A[r, \cdot, \cdot])$. Considering the coefficients of the linear

Table 1. Complexity of MR by KS modeling for cubic system. For different values of n , KS yields a cubic system of n^3 equations in $(3r + 1)n$ variables (assuming $k = n$). The d-reg column gives the degree of regularity for such a semi-regular system without field equations. The complexity column, gives the log base 2 of $\binom{vars+d-1}{d}^{2.8}$.

n	r	vars	eqns	d-reg	cpx
10	10	310	1000	67	699
11	11	374	1331	74	798
12	12	444	1728	81	899
13	13	520	2197	89	1010
14	14	602	2744	97	1123
15	15	690	3375	105	1240

combinations as variables yields a system of $n^2(n - r)$ quadratic equations in $(n - r)r + k$ variables

$$\sum_{j=1}^r \alpha_{ij} A[j, \cdot, \cdot] = A[i, \cdot, \cdot], \text{ for } i = r + 1, \dots, n. \tag{5}$$

Notice that the converse is not necessarily true. A solution to the system in (5) does not necessarily implies the existence of the rank one S_i matrices, neither that A has rank r . However, this is a very overdetermined system, hence a solution is very unlikely, unless indeed A has rank r .

Another approach in the $r < n$ case is to take differentials (or slices) and reduce the problem to a two-dimensional MR problem. If $A \in \mathbb{F}^{n \times n \times n}$ has rank r , the corresponding symmetric trilinear map is likely to have rank r as well. Then, the differentials of this map will have rank less or equal to r . Since the differential operator is lineal, we have an MR problem among the differentials of the symmetric trilinear maps corresponding to M_0, \dots, M_k . In the next section we discuss the relation between the rank of a cubic and its differential in more detail.

To the best of our knowledge, the complexity of solving a system such as (4) has not been studied. It can be seen as a multi-homogeneous system of multi-degree $(1, 1, 1, 1)$, i.e. a tetra-linear system, and assuming some notion of tetra-regularity, analyze it using the techniques in [9]. It should be noticed that the techniques in [9] do not address the semi-regularity inherent to such an overdetermined system. Alternatively, the techniques in [2] could be used to establish the asymptotic behavior of an upper bound of the degree of regularity based on the semi-regularity assumption. Although a complete asymptotic analysis is outside the scope of this paper, Table 1 shows the growth of such bound for selected parameters.

In the case $r \ll n$, the system in (5) has $\mathcal{O}(n^3)$ quadratic equations in $\mathcal{O}(n)$ variables. Since the number of degree two monomials is $\mathcal{O}(n^2)$ the system can be solved by relinearization at degree 2, which reduces to solving a $\mathcal{O}(n^2)$

square matrix. Notice that this is much faster than the KS approach in the two-dimensional case.

3.2 Differentials

Given an instance of the cubic MinRank problem, one can always obtain a quadratic instance by taking the differential (defined below) of the associated polynomials. For example, it is known [14] that computing the differential of the public polynomials of a cubic HFE instance yields an instance of the quadratic HFE scheme, and therefore we can perform a quadratic MinRank attack. In this section we explore the relation between the rank of a cubic polynomial and the rank of its differential. More precisely, given a random homogeneous cubic polynomial $f \in \mathbb{F}[\mathbf{x}]$ of rank r , we want to estimate the rank of the quadratic part of its differential $D_{\mathbf{a}}f(\mathbf{x}) = f(\mathbf{x} + \mathbf{a}) - f(\mathbf{x}) - f(\mathbf{a})$.

The first and principal problem that we face in our analysis is: given an integer r , how can we generate random homogeneous cubic polynomials of rank r ? Or equivalently, how can we generate random symmetric three-dimensional matrices of rank r ? To address these questions, we introduce the concept of symmetric rank. We then choose random polynomials and use Kruskal’s theorem to guarantee that those polynomials have certain symmetric rank.

Definition 4. Let $S \in \mathbb{F}^{n \times n \times n}$ be a three-dimensional symmetric matrix. We define the symmetric rank of S as the minimum number of summands s required to write S as

$$S = \sum_{i=1}^s t_i \mathbf{u}_i \otimes \mathbf{u}_i \otimes \mathbf{u}_i,$$

where $\mathbf{u}_i \in \mathbb{F}^n$, $t_i \in \mathbb{F}$. If such decomposition does not exist, this number is defined to be ∞ . We denote this number by $\text{SRank}(S)$.

The following proposition gives us a sufficient condition over \mathbb{F} to guarantee that for all matrices in $\mathbb{F}^{n \times n \times n}$ the symmetric rank is finite. A more general result is shown in [31, Proposition 7.2].

Proposition 1. Let \mathbb{F} be a finite field with $|\mathbb{F}| \geq 3$. Then each three-dimensional symmetric matrix $S \in \mathbb{F}^{n \times n \times n}$ can be written as $S = \sum_{i=1}^s t_i \mathbf{u}_i \otimes \mathbf{u}_i \otimes \mathbf{u}_i$, where $\mathbf{u}_i \in \mathbb{F}^n$ and $t_i \in \mathbb{F}$.

By the previous proposition, if $|\mathbb{F}| \geq 3$, any homogeneous cubic polynomial f can be written as $\sum_{i=1}^k t_i u_i(\mathbf{x}) u_i(\mathbf{x}) u_i(\mathbf{x})$, where each $u_i(\mathbf{x})$ is a homogeneous linear polynomial and k depends on f . Furthermore, the symmetric rank of a homogeneous cubic $f \in \mathbb{F}[\mathbf{x}]$, denoted by $\text{SRank}(f)$ and defined as the symmetric rank of its symmetric matrix representation, does exist.

The symmetric rank is a good parameter to consider because it is a bound of the rank of the differential.

Proposition 2. Let $f \in \mathbb{F}[\mathbf{x}]$ be a homogeneous cubic polynomial. If g is the quadratic homogeneous part of $Df_{\mathbf{a}}(\mathbf{x})$, then $\text{Rank}(g) \leq \text{SRank}(f)$.

Proof. If f can be written as $f(\mathbf{x}) = \sum_{i=1}^r t_i u_i(\mathbf{x}) u_i(\mathbf{x}) u_i(\mathbf{x})$, then for any $\mathbf{a} \in \mathbb{F}^n$ the quadratic part of $Df_{\mathbf{a}}(\mathbf{x})$ is $\sum_{i=1}^r 3t_i u_i(\mathbf{a}) u_i(\mathbf{x}) u_i(\mathbf{x})$. \square

Let $U = \mathbb{F}^n$. Clearly, each symmetric matrix $A \in \mathbb{F}^{n \times n \times n}$ with symmetric rank r can be written as a sum of exactly r terms of the form $t\mathbf{u} \otimes \mathbf{u} \otimes \mathbf{u}$, where $t \in \mathbb{F} - \{0\}$ and $\mathbf{u} \in U$.

Let \mathcal{S}_r be the function which outputs $A = \sum_{i=1}^r t_i \mathbf{u}_i \otimes \mathbf{u}_i \otimes \mathbf{u}_i$, for $t_i \in \mathbb{F} - \{0\}$ and $\mathbf{u}_i \in U$. By Proposition 1, if $|\mathbb{F}| \geq 3$, then each symmetric matrix $A \in \mathbb{F}^{n \times n \times n}$ with symmetric rank equal to r is in the codomain of \mathcal{S}_r . But some symmetric matrices having symmetric rank less than r can also be there.

The following theorem is a particular case of the known Kruskal’s theorem [19,30]. We use it to argue that if $t_i \in \mathbb{F} - \{0\}$ and $\mathbf{u}_i \in U$ are chosen uniformly at random, then with high probability the corresponding output A of \mathcal{S}_r has symmetric rank equal to r . Moreover, by Kruskal’s theorem with high probability $\text{Rank}(A) = \text{SRank}(A)$. The Kruskal rank of a matrix with columns $\mathbf{u}_1, \dots, \mathbf{u}_m$, denoted by $\text{KRank}(\mathbf{u}_1, \dots, \mathbf{u}_m)$, is defined as the maximum integer k such that any subset of $\{\mathbf{u}_1, \dots, \mathbf{u}_m\}$ of size k is linearly independent.

Theorem 3. *Let \mathbb{F} be a finite field, $\mathbf{u}_1, \dots, \mathbf{u}_r \in U$ and $t_1, \dots, t_r \in \mathbb{F}$. Suppose that $A = \sum_{i=1}^r t_i \mathbf{u}_i \otimes \mathbf{u}_i \otimes \mathbf{u}_i$ and that $2r + 2 \leq \text{KRank}(t_1 \mathbf{u}_1, \dots, t_r \mathbf{u}_r) + 2 \cdot \text{KRank}(\mathbf{u}_1, \dots, \mathbf{u}_r)$. Then $\text{Rank}(A) = r$.*

Suppose $2 \leq r \leq n$. If $\mathbf{u}_1, \dots, \mathbf{u}_r \in U$ are taken uniformly at random, then with high probability a matrix with columns $\mathbf{u}_1, \dots, \mathbf{u}_r$ has full rank. If a matrix with columns $\mathbf{u}_1, \dots, \mathbf{u}_r \in U$ is full rank, then $\text{KRank}(\mathbf{u}_1, \dots, \mathbf{u}_r) = r$ and $\text{KRank}(t_1 \mathbf{u}_1, \dots, t_r \mathbf{u}_r) = r$, for any $t_1, \dots, t_r \in \mathbb{F} - \{0\}$. In this case, by Theorem 3 the corresponding output A of \mathcal{S}_r is such that $\text{Rank}(A) = \text{SRank}(A) = r$.

We experimentally analyze the behavior of the rank of the differential of a polynomial that is the output of $\mathcal{S}_{r,2}$. The experimental results are shown in Fig. 2, where each curve represents the percentage of times that a rank is obtained, over 100,000 iterations.

3.3 Direct Algebraic Attack

The direct algebraic attack, or simply the direct attack, refers to the case when an attacker aims to find the plaintext associated with a ciphertext (c_1, \dots, c_n) directly from the public multivariate equations $p_1 = c_1, \dots, p_n = c_n$, without the knowledge of any other information of the system. In almost all the cases, the most efficient way to perform this attack is to compute a Gröbner basis of the ideal I generated by the multivariate polynomials $p_1 - c_1, \dots, p_n - c_n$.

Gröbner bases have played an important role not only in multivariate cryptography, but also in coding theory and lattices [1,34]. There is a general consensus that when computing a Gröbner basis over a finite field, one of the most efficient ways to do it is to use the F_4 or F_5 algorithms [10,11]. In a recent work [22], the authors used their M4GB algorithm to solve some of Fukuoka’s MQ challenges within 11 days. The complexity of all these algorithms depends on

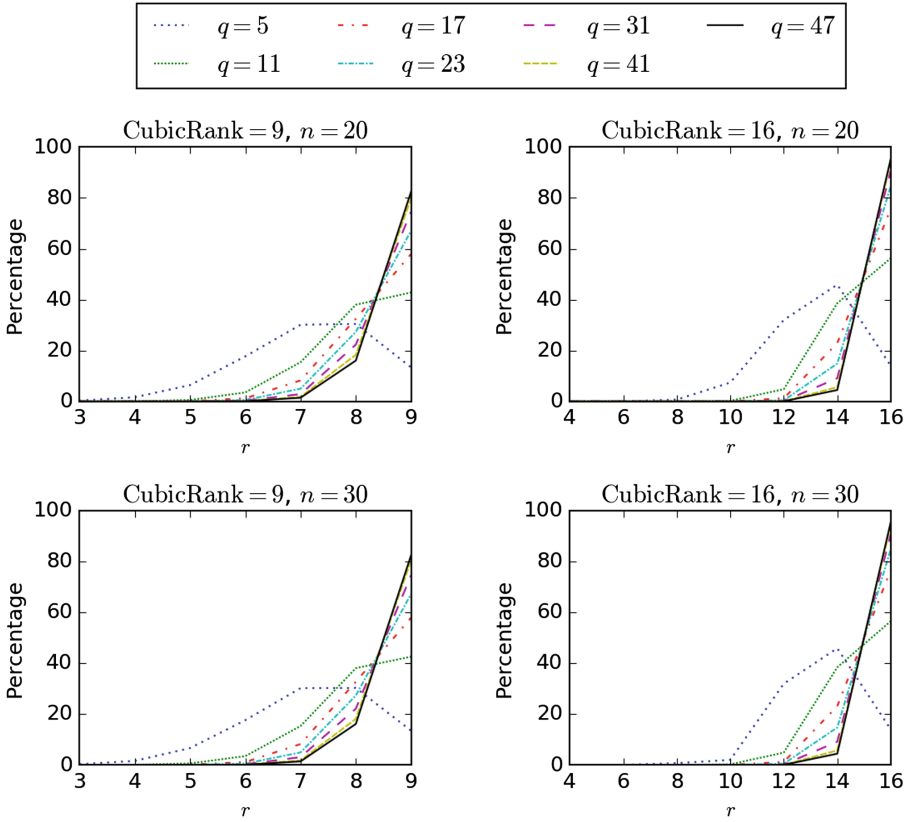


Fig. 2. For different values of q , CubicRank, and n a polynomial f is chosen according $\mathcal{S}_{\text{CubicRank}}$, the $\text{Rank}(Df_{\mathbf{a}})$ is computed for a random $\mathbf{a} \in U$. Each graph represents the percentage of times that a particular value $\text{Rank}(Df_{\mathbf{a}})$ is obtained over 100,000 iterations.

the *degree of regularity* of the system. Since the degree of regularity is hard to determine, it is usually approximated by its *first fall degree*, defined as the first degree at which non-trivial relations between the polynomials p_1, \dots, p_n occur.

Let p be a linear combination of the polynomials p_1, \dots, p_n . We now want to derive an upper bound for the first fall degree $D_{\text{ff}}(p_1, \dots, p_n)$ of the system that depends on $\text{Rank}(p)$. Before we do that, we need the following definition.

Definition 5. The LRank of a homogeneous $\lambda \in \mathbb{F}[x_1, \dots, x_n]$ is the smallest integer s such that there exist linear homogeneous $\mu_1, \dots, \mu_s \in \mathbb{F}[x_1, \dots, x_n]$ with λ contained in the algebra $\mathbb{F}[\mu_1, \dots, \mu_s]$.

Hodges et al. [16] proved that $D_{\text{ff}}(p_1, \dots, p_n)$ is bounded by

$$D_{\text{ff}}(p_1, \dots, p_n) \leq D_{\text{ff}}(p) \leq \frac{\text{LRank}(p)(q - 1) + 5}{2}.$$

Also, since $\text{LRank}(p) \leq 3 \cdot \text{Rank}(p)$ then

$$D_{\text{ff}}(p_1, \dots, p_n) \leq \frac{3 \cdot \text{Rank}(p)(q-1) + 5}{2}. \tag{6}$$

On the other hand, the complexity of finding a Groebner basis \mathcal{G} for the ideal I is bounded by

$$O\left(\binom{n + D_{\text{ff}}}{D_{\text{ff}}}\right)^\omega,$$

where $2 \leq \omega \leq 3$ is the linear algebra constant. When n grows to infinity, the complexity¹ becomes $O(n^{\omega D_{\text{ff}}})$. Therefore, according to the bound in (6), the complexity of finding \mathcal{G} is bounded by

$$O\left(n^{\omega \frac{3 \cdot \text{Rank}(p)(q-1) + 5}{2}}\right).$$

Thus, if q and $\text{Rank}(p)$ are constant, then the complexity of finding \mathcal{G} is polynomial in the number of variables n . We also observe that the complexity is exponential in $\text{Rank}(p)$.

4 Rank Analysis for Cubic Big Field Constructions

As we pointed out in Sect. 2.3, the Big Field Idea has been a basis to propose quadratic multivariate encryption schemes for decades. Nevertheless, Theorem 1 is not restricted to any particular degree, which means that this approach works to generate polynomials of any degree, in particular degree 3. In this section we show that if the central map is a low rank cubic polynomial, then, as in the quadratic case, there must exist a low-rank linear combination of the polynomials of the public key. In particular, we obtain an instance of the cubic MinRank problem which can be solved using the techniques presented in Sect. 3. Thereafter, we discuss the direct algebraic attack on cubic big field schemes having low rank central map.

4.1 Big Field Idea for Cubic Polynomials

Let $\mathcal{F} \in \mathbb{K}[X]$ be a homogeneous weight 3 polynomial given by

$$\mathcal{F}(X) = \sum_{1 \leq i, j, k \leq n} \alpha_{i, j, k} X^{i-1} + X^{j-1} + X^{k-1}$$

and $S, T \in \mathbb{F}^{n \times n}$ invertible matrices. Our first goal is to give an explicit expression for the multivariate cubic polynomials of the composition $T \circ \phi \circ \mathcal{F} \circ \phi^{-1} \circ S$. We begin by representing the map \mathcal{F} as $\mathcal{F}(X) = \mathcal{T}(\mathbf{X}, \mathbf{X}, \mathbf{X})$ where $\mathbf{X} =$

¹ Notice that we are using an upper bound to estimate the complexity. This is a customary usage for this kind of attacks. In practice, it has been observed [32] that, on average, this bound is not too far from the actual complexity.

$(X^{q^0}, \dots, X^{q^{n-1}})^\top$ and $\mathcal{T} : \mathbb{K}^n \times \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}$ is the trilinear form given by

$$\mathcal{T}(\boldsymbol{\beta}, \boldsymbol{\delta}, \boldsymbol{\gamma}) = \sum_{1 \leq i, j, k \leq n} \alpha_{i, j, k} \cdot (\beta_i \delta_j \gamma_k).$$

Let A be the three-dimensional matrix whose entry (i, j, k) is given by $\alpha_{i, j, k}$, and assume without loss of generality that the matrix A is symmetric (otherwise we can take the matrix whose (i, j, k) entry is given by $\frac{1}{3!} \cdot (A[i, j, k] + A[i, k, j] + A[j, i, k] + A[j, k, i] + A[k, i, j] + A[k, j, i])$, which represents the same trilinear form \mathcal{T}).

Let $\mathcal{T}' : \mathbb{K}^n \times \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}$ be the trilinear form given by $\mathcal{T}'(\boldsymbol{\beta}, \boldsymbol{\delta}, \boldsymbol{\gamma}) = \mathcal{T}(\Delta S \boldsymbol{\beta}, \Delta S \boldsymbol{\delta}, \Delta S \boldsymbol{\gamma})$, we can write this trilinear form as

$$\mathcal{T}'(\boldsymbol{\beta}, \boldsymbol{\delta}, \boldsymbol{\gamma}) = \sum_{1 \leq i, j, k \leq n} \alpha'_{i, j, k} \cdot (\beta_i \delta_j \gamma_k)$$

where $\alpha'_{i, j, k} = \mathcal{T}'(\mathbf{e}_i, \mathbf{e}_j, \mathbf{e}_k) = \mathcal{T}(\Delta S \mathbf{e}_i, \Delta S \mathbf{e}_j, \Delta S \mathbf{e}_k)$.

Let A' be the three-dimensional matrix whose entry (i, j, k) is given by $\alpha'_{i, j, k}$. Notice that this is the cubic version of the matrix $(\Delta S)^\top A (\Delta S)$ from Sect. 2.3. It is easy to see that the matrix A' is symmetric since the matrix A is.

Let $\mathbf{a} \in \mathbb{F}^n$ and let $\alpha = \phi^{-1}(S\mathbf{a})$, we know that $\text{Fr}(\alpha) = \Delta \cdot \phi(\alpha) = \Delta S \cdot \mathbf{a}$ and therefore

$$\begin{aligned} \mathcal{F} \circ \phi^{-1}(S\mathbf{a}) &= \mathcal{F}(\alpha) = \mathcal{T}(\text{Fr}(\alpha), \text{Fr}(\alpha), \text{Fr}(\alpha)) = \mathcal{T}(\Delta S \cdot \mathbf{a}, \Delta S \cdot \mathbf{a}, \Delta S \cdot \mathbf{a}) \\ &= \mathcal{T}'(\mathbf{a}, \mathbf{a}, \mathbf{a}) = \sum_{1 \leq i, j, k \leq n} \alpha'_{i, j, k} \cdot (a_i a_j a_k). \end{aligned}$$

Let $R_1, \dots, R_n \in \mathbb{F}^{n \times n \times n}$ be three-dimensional symmetric matrices such that $A' = y^0 \cdot R_1 + y^1 \cdot R_2 + \dots + y^{n-1} \cdot R_n$, where $y^0, y^1 \dots y^{n-1}$ is the basis of \mathbb{K} over \mathbb{F} , as explained in Sect. 2.3. Then

$$\begin{aligned} \mathcal{F} \circ \phi^{-1} \circ S(\mathbf{a}) &= \sum_{1 \leq i, j, k \leq n} \alpha'_{i, j, k} \cdot (a_i a_j a_k) \\ &= \sum_{1 \leq i, j, k \leq n} \left(\sum_{\ell=1}^n y^{\ell-1} R_\ell[i, j, k] \right) \cdot (a_i a_j a_k) \\ &= \sum_{\ell=1}^n y^{\ell-1} \underbrace{\left(\sum_{1 \leq i, j, k \leq n} R_\ell[i, j, k] \cdot (a_i a_j a_k) \right)}_{t_\ell}. \end{aligned}$$

Since $t_\ell \in \mathbb{F}$, we obtain that $\phi \circ \mathcal{F} \circ \phi^{-1} \circ S(\mathbf{a}) = (t_1, \dots, t_\ell)^\top$. Therefore, each cubic polynomial in the composition $\phi \circ \mathcal{F} \circ \phi^{-1} \circ S$ is given by $f_\ell(\mathbf{x}) = \sum_{1 \leq i, j, k \leq n} R_\ell[i, j, k] \cdot (x_i x_j x_k)$. Finally, when we apply the transformation T we obtain that each cubic polynomial in the composition $P = T \circ \phi \circ \mathcal{F} \circ \phi^{-1} \circ S$ is given by

$$p_\ell(\mathbf{x}) = \sum_{1 \leq i, j, k \leq n} \left(\sum_{t=1}^n T[\ell, t] \cdot R_t[i, j, k] \right) \cdot (x_i x_j x_k).$$

As a conclusion, if we let A_ℓ be the matrix whose entry (i, j, k) is given by $\sum_{t=1}^n T[\ell, t] \cdot R_t[i, j, k]$ then we obtain that this is the symmetric matrix corresponding to the ℓ -th polynomial in P . In particular, this shows we can compute efficiently the composition $T \circ \phi \circ \mathcal{F} \circ \phi^{-1} \circ S$ from S, T and \mathcal{F} .

4.2 Existence of Low Rank Linear Combination

Let us continue with the same setting as before, and let r be the rank of A , which in particular means that A can be written as $\sum_{\ell=1}^r \mathbf{u}_\ell \otimes \mathbf{v}_\ell \otimes \mathbf{w}_\ell$. Suppose that r is small. In this section we prove that there exists a low-rank linear combination of the three-dimensional matrices representing the composition P , and in Sect. 3.1 we showed how to find such combination.

Recall that the matrix A' was defined as $A'[i, j, k] = \mathcal{T}(\Delta S \mathbf{e}_i, \Delta S \mathbf{e}_j, \Delta S \mathbf{e}_k)$, we claim that the rank of this matrix is at most the rank of A . We show this by exhibiting vectors $\mathbf{u}'_\ell, \mathbf{v}'_\ell, \mathbf{w}'_\ell \in \mathbb{K}^n$ such that $A' = \sum_{\ell=1}^r \mathbf{u}'_\ell \otimes \mathbf{v}'_\ell \otimes \mathbf{w}'_\ell$. Let M be the matrix ΔS , we define $\mathbf{u}'_\ell = \sum_{i=1}^n \mathbf{u}_\ell[i] \cdot M[i, \cdot]$, $\mathbf{v}'_\ell = \sum_{i=1}^n \mathbf{v}_\ell[i] \cdot M[i, \cdot]$ and $\mathbf{w}'_\ell = \sum_{i=1}^n \mathbf{w}_\ell[i] \cdot M[i, \cdot]$, then

$$\begin{aligned} &A'[i', j', k'] \\ &= \mathcal{T}'(M \mathbf{e}_{i'}, M \mathbf{e}_{j'}, M \mathbf{e}_{k'}) \\ &= \sum_{1 \leq i, j, k \leq n} A[i, j, k] \cdot ((M \mathbf{e}_{i'})[i] \cdot (M \mathbf{e}_{j'})[j] \cdot (M \mathbf{e}_{k'})[k]) \\ &= \sum_{1 \leq i, j, k \leq n} \left(\sum_{\ell=1}^r \mathbf{u}_\ell[i] \cdot \mathbf{v}_\ell[j] \cdot \mathbf{w}_\ell[k] \right) ((M[i, \cdot] \mathbf{e}_{i'}) \cdot (M[j, \cdot] \mathbf{e}_{j'}) \cdot (M[k, \cdot] \mathbf{e}_{k'})) \\ &= \sum_{\ell=1}^r \sum_{1 \leq i, j, k \leq n} (\mathbf{u}_\ell[i] M[i, \cdot] \mathbf{e}_{i'}) (\mathbf{v}_\ell[j] M[j, \cdot] \mathbf{e}_{j'}) (\mathbf{w}_\ell[k] M[k, \cdot] \mathbf{e}_{k'}) \\ &= \sum_{\ell=1}^r \left(\sum_{i=1}^n \mathbf{u}_\ell[i] M[i, \cdot] \mathbf{e}_{i'} \right) \left(\sum_{j=1}^n \mathbf{v}_\ell[j] M[j, \cdot] \mathbf{e}_{j'} \right) \left(\sum_{k=1}^n \mathbf{w}_\ell[k] M[k, \cdot] \mathbf{e}_{k'} \right) \\ &= \sum_{\ell=1}^r [(\mathbf{u}'_\ell) \mathbf{e}_{i'}] [(\mathbf{v}'_\ell) \mathbf{e}_{j'}] [(\mathbf{w}'_\ell) \mathbf{e}_{k'}] \\ &= \sum_{\ell=1}^r \mathbf{u}'_\ell[i'] \cdot \mathbf{v}'_\ell[j'] \cdot \mathbf{w}'_\ell[k']. \end{aligned}$$

From this we conclude that $A' = \sum_{\ell=1}^r \mathbf{u}'_\ell \otimes \mathbf{v}'_\ell \otimes \mathbf{w}'_\ell$ and hence $\text{Rank}(A') \leq r$. Now let $(\lambda_1, \dots, \lambda_n) = (y^0, \dots, y^{n-1}) \cdot T^{-1}$, then

$$\sum_{i=1}^n \lambda_i A_i = \sum_{i=1}^n \lambda_i \left(\sum_{j=1}^n T[i, j] \cdot R_j \right) = \sum_{j=1}^n R_j \sum_{i=1}^n T[i, j] \cdot \lambda_i = \sum_{j=1}^n R_j \cdot y^{j-1} = A'.$$

This shows that there is a linear combination of the matrices representing the public key whose result is a low rank three-dimensional matrix. This yields directly an instance of the cubic MinRank problem which can be solved with the extension of the Kipnis-Shamir modeling presented in Sect. 3. As we mentioned before, this is by itself a weakness of the scheme, as it allows distinguishing public keys from random polynomial systems and also have implications on the degree of regularity of the system, as stated in Sect. 3.3. Moreover, the coefficients we have obtained here carry the same information about the secret key as those in the original (quadratic) MinRank attack, and this can be used in a similar way to construct equivalent keys.

4.3 Algebraic Attack for Cubic Big Field Constructions

As a complement of Sect. 3.3, we now consider the case when the polynomials p_1, \dots, p_n are constructed using the big field idea for cubic polynomials. Hodges et al. [16] proved that for a scheme with core polynomial of weight 3, its first fall degree $D_{\text{ff}}(p_1, \dots, p_n)$ is bounded by

$$D_{\text{ff}}(p_1, \dots, p_n) \leq \frac{\text{LRank}(P_0)(q - 1) + 5}{2}.$$

Here P_0 is the homogeneous part of highest degree of the core polynomial \mathcal{F} seen as an element of the graded algebra $\mathbb{K}[X_0, \dots, X_{n-1}]/(X_0^q, \dots, X_{n-1}^q)$, where X_i corresponds to X^{q^i} , for $i = 0, \dots, n - 1$. In our case

$$P_0 = \sum_{1 \leq i, j, k \leq n} \alpha_{i, j, k} X_{i-1} X_{j-1} X_{k-1}.$$

If we take α_{ijk} uniformly at random, then with high probability $\text{LRank}(P_0) \leq \text{Rank}(P_0)$, so

$$D_{\text{ff}}(p_1, \dots, p_n) \leq \frac{\text{Rank}(\mathcal{F})(q - 1) + 5}{2}, \tag{7}$$

since $\text{Rank}(P_0) = \text{Rank}(\mathcal{F})$.

In [16] the authors show that if $\deg \mathcal{F} = D$, then $\text{LRank}(\mathcal{F}) \leq \lfloor \log_q(D - 2) \rfloor + 1$, and hence

$$D_{\text{ff}}(p_1, \dots, p_n) \leq \frac{(q - 1) \lfloor \log_q(D - 2) \rfloor + 4 + q}{2}. \tag{8}$$

We now want to experimentally study the tightness of the bound (8), as they did in [16] for different parameters². In Table 2 we present some of the results obtained for different values of the parameters q, n and t , where t is the smallest integer such that $D \leq q^t - 1$. The value B corresponds to the bound given by Eq. (8), and D_{ff} is the first fall degree of the system for each choice of the parameters, which is read from Magma’s verbose output. All the polynomials used in the attack were built as it will be explained in Sect. 4.1, and for all cases we have included the field equations, i.e., $x_i^q - x_i$ for $i = 1, \dots, n$.

² Table 1 in [16] do not include the values for the parameters we are interested in, so we constructed our own version of it.

Table 2. Experimental results to study the tightness of the bound for D_{ff} given by (8), for different choices of the parameters q , t and n . The value of D_{ff} is read from Magma’s verbose output.

q	t	n	B	D_{ff}	q	t	n	B	D_{ff}	q	t	n	B	D_{ff}	q	t	n	B	D_{ff}
5	3	8	8	8	7	3	8	11	10	11	3	8	17	13	17	3	8	26	17
5	3	9	8	8	7	3	9	11	10	11	3	9	17	14	17	3	9	26	18
5	3	10	8	8	7	3	10	11	10	11	3	10	17	15	17	3	10	26	18
5	4	8	10	9	7	4	8	14	10	11	4	8	22	13	17	4	8	34	17
5	4	9	10	9	7	4	9	14	11	11	4	9	22	14	17	4	9	34	18
5	4	10	10	10	7	4	10	14	12	11	4	10	22	15	17	4	10	34	18
5	5	8	12	9	7	5	8	17	10	11	5	8	27	13	17	5	8	42	17
5	5	9	12	9	7	5	9	17	11	11	5	9	27	14	17	5	9	42	18
5	5	10	12	10	7	5	10	17	12	11	5	10	27	15	17	5	10	42	18

We notice that the bound given by (8) is very tight for small values of q and t , and that it starts to widen considerably as q increases, and with a smaller pace as t increases. We also observe that for fixed q and t , the bound gets tighter as n increases. It is very clear that the bound needs to be improved for larger values of q .

5 Conclusions and Future Work

The minimum rank of a linear combination of the public polynomials is an important property of multivariate schemes. We have shown that this is still true for cubic schemes. The rank for cubic maps can be directly studied and exploited.

Many attacks have shown that it is hard to escape a low-rank when constructing quadratic encryption schemes. A high rank defect is necessary to allow decryption, leaving a low rank map exposed. Our rank analysis of cubic cryptosystems shows that low, fixed rank constructions have no chance of being secure. On the other hand, we are convinced that cubic polynomials allow more versatile constructions than quadratic, where a rank defect can help decryption but leave a rank large enough so that it does not necessarily represent a weakness.

This work is preliminary in the sense that it opens new questions. Can we construct cubic maps with a rank defect that allows decryption but leave a rank large enough for security? Other algorithms to solve the cubic-min-rank problem are likely, for example based on the minors modeling or on guessing kernel vectors. The complexity of each of these approaches needs to be studied more carefully (even in the quadratic case). These attacks could also be extendable to the cases where the field has characteristic 2 or 3. Finally, the hardness of rank problems for three-dimensional matrices can be further harvest as a security assumption.

Acknowledgements. This work was partially supported by “Fondo Nacional de Financiamiento para la Ciencia, la Tecnología y la Innovación Francisco José de Caldas”, Colciencias (Colombia), Project No. 111865842333 and Contract No. 049-2015.

We would like to thank Jintai Ding for very useful discussions. We would also like to thank the reviewers of PQCrypto 2018 for their constructive reviews and suggestions. And last but not least, we thank the Facultad de Ciencias of the Universidad Nacional de Colombia sede Medellín for granting us access to the Enlace server, where we ran most of the experiments of this paper.

References

1. Aliasgari, M., Sadeghi, M.R., Panario, D.: Gröbner bases for lattices and an algebraic decoding algorithm. In: 2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 1414–1415, September 2011
2. Bardet, M., Faugère, J.-C., Salvy, B., Yang, B.-Y.: Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In: Eighth International Symposium on Effective Methods in Algebraic Geometry, MEGA 2005, pp. 1–14 (2005)
3. Bettale, L., Faugère, J.-C., Perret, L.: Cryptanalysis of HFE, multi-HFE and variants for odd and even characteristic. *Des. Codes Crypt.* **69**(1), 1–52 (2013)
4. Buss, J.F., Frandsen, G.S., Shallit, J.O.: The computational complexity of some problems of linear algebra. *J. Comput. Syst. Sci.* **58**(3), 572–596 (1999)
5. Chen, C.-H.O., Chen, M.-S., Ding, J., Werner, F., Yang, B.-Y.: Odd-char multivariate hidden field equations. *IACR Cryptology ePrint Archive*, 2008:543 (2008)
6. Ding, J., Hodges, T.J.: Inverting HFE systems is quasi-polynomial for all fields. In: Rogaway, P. (ed.) *CRYPTO 2011*. LNCS, vol. 6841, pp. 724–742. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_41
7. Ding, J., Petzoldt, A., Wang, L.: The cubic simple matrix encryption scheme. In: Mosca, M. (ed.) *PQCrypto 2014*. LNCS, vol. 8772, pp. 76–87. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_5
8. Escudero, D.: Groebner bases and applications to the security of multivariate public key cryptosystems (2016). <http://cs.au.dk/~escudero/files/TDG.pdf>. Accessed 25 Nov 2017
9. Faugère, J.-C., El Din, M.S., 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)
10. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases (F_4). *J. Pure Appl. Algebra* **139**(1-3), 61–88 (1999). (Effective methods in algebraic geometry, Saint-Malo (1998))
11. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (f5). In: *Proceedings of 2002 International Symposium on Symbolic and Algebraic Computation, ISSAC 2002*, pp. 75–83. ACM, New York (2002)
12. Faugère, J.-C., Levy-dit-Vehel, F., Perret, L.: Cryptanalysis of MinRank. In: Wagner, D. (ed.) *CRYPTO 2008*. LNCS, vol. 5157, pp. 280–296. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_16
13. Goubin, L., Courtois, N.T.: Cryptanalysis of the TTM cryptosystem. In: Okamoto, T. (ed.) *ASIACRYPT 2000*. LNCS, vol. 1976, pp. 44–57. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_4

14. Hashimoto, Y.: Multivariate public key cryptosystems. In: Takagi, T., Wakayama, M., Tanaka, K., Kunihiro, N., Kimoto, K., Duong, D. (eds.) *Mathematical Modelling for Next-Generation Cryptography*. Mathematics for Industry, vol. 29, pp. 17–42. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-5065-7_2
15. Hillar, C.J., Lim, L.-H.: Most tensor problems are NP-hard. *J. ACM* **60**(6), 45:1–45:39 (2013)
16. Hodges, T.J., Petit, C., Schlather, J.: First fall degree and weil descent. *Finite Fields Appl.* **30**, 155–177 (2014)
17. Howell, T.D.: Global properties of tensor rank. *Linear Algebra Appl.* **22**(Suppl. C), 9–23 (1978)
18. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE public key cryptosystem by relinearization. In: Wiener, M. (ed.) *CRYPTO 1999*. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_2
19. Kruskal, J.B.: Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra Appl.* **18**(2), 95–138 (1977)
20. Landsberg, J.M.: *Tensors: Geometry and Applications*. Graduate Studies in Mathematics, vol. 128. American Mathematical Society, Providence (2012)
21. Lidl, R., Niederreiter, H.: *Finite Fields*. Encyclopedia of Mathematics and Its Applications, 2nd edn, vol. 20. Cambridge University Press, Cambridge (1997). With a foreword by P.M. Cohn
22. Makarim, R.H., Stevens, M.: M4GB: an efficient Gröbner-basis algorithm. In: *Proceedings of 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC 2017, pp. 293–300. ACM, New York (2017)
23. Matsumoto, T., Imai, H.: Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In: Barstow, D., et al. (eds.) *EUROCRYPT 1988*. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-45961-8_39
24. Moody, D., Perlner, R., Smith-Tone, D.: An asymptotically optimal structural attack on the ABC multivariate encryption scheme. In: Mosca, M. (ed.) *PQCrypto 2014*. LNCS, vol. 8772, pp. 180–196. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_11
25. Moody, D., Perlner, R., Smith-Tone, D.: Improved attacks for characteristic-2 parameters of the cubic ABC simple matrix encryption scheme. In: Lange, T., Takagi, T. (eds.) *PQCrypto 2017*. LNCS, vol. 10346, pp. 255–271. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_15
26. Moody, D., Perlner, R., Smith-Tone, D.: Key recovery attack on the cubic ABC simple matrix multivariate encryption scheme. In: Avanzi, R., Heys, H. (eds.) *SAC 2016*. LNCS, vol. 10532, pp. 543–558. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_29
27. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U. (ed.) *EUROCRYPT 1996*. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_4
28. Patarin, J., Courtois, N., Goubin, L.: QUARTZ, 128-bit long digital signatures. In: Naccache, D. (ed.) *CT-RSA 2001*. LNCS, vol. 2020, pp. 282–297. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45353-9_21
29. Porras, J., Baena, J., Ding, J.: ZHFE, a new multivariate public key encryption scheme. In: Mosca, M. (ed.) *PQCrypto 2014*. LNCS, vol. 8772, pp. 229–245. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_14

30. Shmuel, F.: Remarks on the symmetric rank of symmetric tensors, January 2016. [arXiv.org/pdf/1505.00860](https://arxiv.org/pdf/1505.00860)
31. Shmuel, F., Stawiska, M.: Best approximation on semi-algebraic sets and k -border rank approximation of symmetric tensors, November 2013. [arXiv.org/pdf/1311.1561](https://arxiv.org/pdf/1311.1561)
32. Spaenlehauer, P.-J.: Solving multi-homogeneous and determinantal systems. Algorithms - Complexity - Applications. Ph.D. thesis, Université Paris 6 (2012)
33. Yang, B.-Y., Chen, J.-M.: Building secure tame-like multivariate public-key cryptosystems: the new TTS. In: Boyd, C., González Nieto, J.M. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 518–531. Springer, Heidelberg (2005). https://doi.org/10.1007/11506157_43
34. Barrientos, I.Á., Borges-Quintana, M., Borges-Trenard, M.A., Panario, D.: Computing Gröbner bases associated with lattices. Adv. Math. Commun. **10**(4), 851–860 (2016)



Improved Cryptanalysis of HFEv- via Projection

Jintai Ding¹, Ray Perlner², Albrecht Petzoldt², and Daniel Smith-Tone^{2,3}(✉)

¹ Department of Mathematical Sciences, University of Cincinnati,
Cincinnati, OH, USA
jintai.ding@uc.edu

² National Institute of Standards and Technology, Gaithersburg, MD, USA
{ray.perlner,daniel.smith}@nist.gov, albrecht.petzoldt@gmail.com

³ Department of Mathematics, University of Louisville, Louisville, KY, USA

Abstract. The HFEv- signature scheme is one of the most studied multivariate schemes and one of the major candidates for the upcoming standardization of post-quantum digital signature schemes. In this paper, we propose three new attack strategies against HFEv-, each of them using the idea of projection. Especially our third attack is very effective and is, for some parameter sets, the most efficient known attack against HFEv-. Furthermore, our attack requires much less memory than direct and rank attacks. By our work, we therefore give new insights in the security of the HFEv- signature scheme and restrictions for the parameter choice of a possible future standardized HFEv- instance.

Keywords: Multivariate cryptography · HFEv- · MinRank
Gröbner basis · Projection

1 Introduction

Multivariate cryptography is one of the main candidates for establishing cryptosystems which resist attacks with quantum computers (so called post-quantum cryptosystems). Especially in the area of digital signatures, there exists a large number of practical multivariate schemes such as UOV [1] and Rainbow [2].

Another well known multivariate signature scheme is the HFEv- signature scheme, which was first proposed by Patarin et al. in [3]. Most notably about this scheme are its very short signatures, which are currently the shortest signatures of all existing schemes (both classical and post-quantum).

In this paper we propose three new attacks against the HFEv- signature scheme, each of them using the idea of projection. This means that each of our attacks reduces the number of variables in the system by guessing, either before or after the attack itself.

The rights of this work are transferred to the extent transferable according to title 17 § 105 U.S.C.

The most interesting results hereby are provided by a distinguishing based attack, which is related to the hybrid approach of the direct attack [4]. The goal of our attack is to remove the vinegar modifier. This allows the attacker to follow up with any key recovery or signature forgery attack applicable to an HFE-instance with the same degree bound and the same number of removed equations as the original HFEv- instance. The attack is very effective and outperforms, for selected parameter sets, all other attacks against HFEv-. Furthermore, the memory requirements of our attack are far less than those of direct and MinRank attacks.

The rest of the paper is organized as follows. In Sect. 2, we give a short overview of multivariate cryptography and introduce the HFEv- cryptosystem, while Sect. 3 reviews the previous cryptanalysis of this scheme. Section 4 describes our first two attacks, which combine the MinRank attack with the idea of projection. In Sect. 5, we present then our distinguishing based attack, whose complexity is analyzed in Sect. 6. Finally, Sect. 7 discusses shortly ideas for future work.

2 Hidden Field Equations

2.1 Multivariate Cryptography

The basic objects of multivariate cryptography are systems of multivariate quadratic polynomials over a finite field \mathbb{F} . The security of multivariate schemes is based on the *MQ Problem* of solving such a system. The MQ Problem is proven to be NP-Hard even for quadratic polynomials over the field $\text{GF}(2)$ [5] and believed to be hard on average (both for classical and quantum computers).

To build a multivariate public key cryptosystem (MPKC), one starts with an easily invertible quadratic map $\mathcal{F} : \mathbb{F}^n \rightarrow \mathbb{F}^m$ (*central map*). To hide the structure of \mathcal{F} in the public key, we compose it with two invertible affine (or linear) maps $\mathcal{T} : \mathbb{F}^m \rightarrow \mathbb{F}^m$ and $\mathcal{U} : \mathbb{F}^n \rightarrow \mathbb{F}^n$. The *public key* of the scheme is therefore given by $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{U} : \mathbb{F}^n \rightarrow \mathbb{F}^m$. The relation between the easily invertible central map \mathcal{F} and the public key \mathcal{P} is referred to as a morphism of polynomials.

The *private key* consists of the three maps \mathcal{T} , \mathcal{F} and \mathcal{U} and therefore allows to invert the public key. To generate a signature for a document (hash value) $\mathbf{h} \in \mathbb{F}^m$, one computes recursively $\mathbf{x} = \mathcal{T}^{-1}(\mathbf{h}) \in \mathbb{F}^m$, $\mathbf{y} = \mathcal{F}^{-1}(\mathbf{x}) \in \mathbb{F}^n$ and $\mathbf{z} = \mathcal{U}^{-1}(\mathbf{y}) \in \mathbb{F}^n$. To check the authenticity of a signature $\mathbf{z} \in \mathbb{F}^n$, one simply computes $\mathbf{h}' = \mathcal{P}(\mathbf{z}) \in \mathbb{F}^m$. If the result is equal to \mathbf{h} , the signature is accepted, otherwise rejected. This process is illustrated in Fig. 1.

2.2 HFE Variants

The HFE encryption scheme was proposed by Patarin in [6]. The scheme belongs to the *BigField* family of multivariate schemes, which means that it uses a degree n extension field \mathbb{E} of \mathbb{F} as well as an isomorphism $\phi : \mathbb{F}^n \rightarrow \mathbb{E}$. The central map is a univariate polynomial map over \mathbb{E} of the form

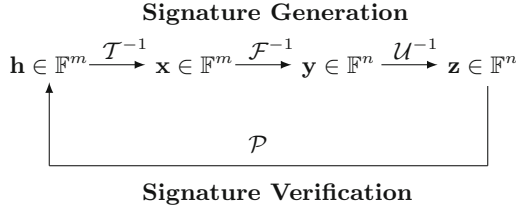


Fig. 1. Signature generation and verification for multivariate signature schemes

$$\mathcal{F}(X) = \sum_{0 \leq i, j}^{q^i + q^j \leq D} \alpha_{ij} X^{q^i + q^j} + \sum_{i=0}^{q^i \leq D} \beta_i X^{q^i} + \gamma.$$

Due to the special structure of \mathcal{F} , the map $\bar{\mathcal{F}} = \phi^{-1} \circ \mathcal{F} \circ \phi$ is a quadratic map over the vector space \mathbb{F}^n . In order to hide the structure of \mathcal{F} in the public key, $\bar{\mathcal{F}}$ is composed with two affine maps \mathcal{T} and \mathcal{U} , i.e. $\mathcal{P} = \mathcal{T} \circ \bar{\mathcal{F}} \circ \mathcal{U}$.

After the basic scheme was broken by direct [7] and rank attacks [8], several versions of HFE for digital signatures have been proposed. Basically, these schemes use two different techniques: the minus and the vinegar modification. For the HFEEv- signature scheme [3], the central map \mathcal{F} has the form

$$\mathcal{F}(X, \bar{x}_V) = \sum_{0 \leq i, j}^{q^i + q^j \leq D} \alpha_{ij} X^{q^i + q^j} + \sum_{i=0}^{q^i \leq D} \beta_i(x_{n+1}, \dots, x_{n+v}) X^{q^i} + \gamma(x_{n+1}, \dots, x_{n+v}),$$

where β_i and γ are linear and quadratic maps in the vinegar variables $\bar{x}_V = (x_{n+1}, \dots, x_{n+v})$ respectively. Defining $\psi : \mathbb{F}^{n+v} \rightarrow \mathbb{E} \times \mathbb{F}^v$ by $\psi = \phi \times id_v$, the public key has the form

$$\mathcal{P} = \mathcal{T} \circ \phi^{-1} \circ \mathcal{F} \circ \psi \circ \mathcal{U} : \mathbb{F}^{n+v} \rightarrow \mathbb{F}^{n-a}$$

with two affine maps $\mathcal{T} : \mathbb{F}^n \rightarrow \mathbb{F}^{n-a}$ and $\mathcal{U} : \mathbb{F}^{n+v} \rightarrow \mathbb{F}^{n+v}$, and is a multivariate quadratic map with coefficients and variables over \mathbb{F} .

Signature Generation: To generate a signature \mathbf{z} for a document d , one uses a hash function $\mathcal{H} : \{0, 1\}^* \rightarrow \mathbb{F}^{n-a}$ to compute a hash value $\mathbf{h} = \mathcal{H}(d) \in \mathbb{F}^{n-a}$ and performs the following four steps

1. Compute a preimage $\mathbf{x} \in \mathbb{F}^n$ of \mathbf{h} under the affine map \mathcal{T} and set $X = \phi(\mathbf{x}) \in \mathbb{E}$.
2. Choose random values for the vinegar variables x_{n+1}, \dots, x_{n+v} and substitute them into the central map to obtain the parametrized map \mathcal{F}_V .
3. Solve the univariate polynomial equation $\mathcal{F}_V(Y) = X$ over the extension field \mathbb{E} by Berlekamp’s algorithm.
4. Compute the signature $\mathbf{z} = \mathcal{U}^{-1}(\phi^{-1}(Y) || x_{n+1} || \dots || x_{n+v}) \in \mathbb{F}^{n+v}$.

Signature Verification: To check the authenticity of a signature $\mathbf{z} \in \mathbb{F}^{n+v}$, the verifier computes $\mathbf{h} = \mathcal{H}(d)$ and $\mathbf{h}' = \mathcal{P}(\mathbf{z})$. If $\mathbf{h}' = \mathbf{h}$ holds, the signature is accepted, otherwise rejected.

3 Previous Cryptanalysis

3.1 Direct Algebraic Attack

The direct algebraic attack is the most straightforward way to attack a multivariate cryptosystem such as HFEv-. In this attack, one considers the public equation $\mathcal{P}(\mathbf{z}) = \mathbf{h}$ as an instance of the MQ-Problem. In the case of HFEv-, this public system is slightly underdetermined. In order to make the solution space zero dimensional, one therefore fixes $a + v$ variables in order to get a determined system before applying an algorithm like XL [9] or a Gröbner basis method such as F_4 or F_5 [10,11]. In some cases one gets better results by guessing additional variables, even if this requires to run the Gröbner basis algorithm several times (hybrid approach [4]).

The complexity of a direct attack using the hybrid approach against a system of m quadratic equations in n variables can be estimated as

$$Comp_{direct} = \min_k q^k \cdot 3 \cdot \binom{n - k + d_{reg}}{d_{reg}}^2 \cdot \binom{n - k}{2},$$

where d_{reg} is the so called *degree of regularity* of the multivariate system. Note that this formula gives only a rough estimate and lower bound of the complexity of a direct attack, since it assumes that the linear systems appearing during the attack are very sparse systems. It is not clear if this assumption holds and if the used Wiedemann algorithm can work with the assumed complexity.

Experiments have shown that the public systems of HFE and its variants can be solved significantly faster than random systems [7,12]. This phenomenon was studied by Ding et al. in a series of papers [13–15]. In [15] it was shown that the degree of regularity of solving an HFEv- system is upper bounded by

$$d_{reg, HFEv-} \leq \begin{cases} \frac{(q-1) \cdot (r+a+v-1)}{2} + 2 & q \text{ even and } r + a \text{ odd} \\ \frac{(q-1) \cdot (r+a+v)}{2} + 2 & \text{otherwise} \end{cases}. \quad (1)$$

3.2 MinRank

The historically most effective attack on the HFE family of cryptosystems is the MinRank attack which exploits the algebraic consequence of a low degree bound D . This low degree bound leads to the fact that the central map has a low Q-rank.

Definition 1. *The Q-rank of a multivariate quadratic map $\mathcal{F} : \mathbb{F}^n \rightarrow \mathbb{F}^n$ over the finite field \mathbb{F} with q elements is the rank of the quadratic form \mathcal{Q} on $\mathbb{E}[X_1, \dots, X_n]$ defined by $\mathcal{Q}(X_1, \dots, X_n) = \phi \circ \mathcal{F} \circ \phi^{-1}(X)$, under the identification $X_1 = X, X_2 = X^q, \dots, X_n = X^{q^{n-1}}$.*

Q-rank is invariant under one-sided isomorphisms of polynomials of the form $\mathcal{G} = \mathcal{I} \circ \mathcal{F} \circ \mathcal{U}$, where \mathcal{I} is the identity transformation. Q-rank is not, however, invariant under isomorphisms of polynomials in general. The min-Q-Rank of a

quadratic map \mathcal{F} is the minimum Q-rank of any quadratic map in the isomorphism class of \mathcal{F} . This quantity is invariant under isomorphisms of polynomials, and is the relevant quantity for cryptanalysis. For historical reasons, language is often abused and the term Q-rank is used in place of min-Q-rank.

As an example, consider an odd characteristic instance of HFE. We may write the homogeneous quadratic part of F as

$$\begin{bmatrix} X & X^q & \cdots & X^{q^{n-1}} \end{bmatrix} \begin{bmatrix} \alpha_{1,1} & \alpha'_{1,2} & \cdots & \alpha'_{1,d} & 0 & \cdots & 0 \\ \alpha'_{1,2} & \alpha_{2,2} & \cdots & \alpha'_{2,d} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha'_{1,d} & \alpha'_{2,d} & \cdots & \alpha_{d,d} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} X \\ X^q \\ \vdots \\ X^{q^{n-1}} \end{bmatrix},$$

where $\alpha'_{i,j} = \frac{1}{2}\alpha_{i,j}$ and $d = \lceil \log_q(D) \rceil$. Clearly, this quadratic form over the ring $\mathbb{E}[X_1, \dots, X_n]$ has rank d , and thus the HFE central map has Q-rank d .

The first iteration of the MinRank attack in the *BigField* setting is the Kipnis-Shamir (KS) attack of [8]. Via polynomial interpolation, the public key can be expressed as a quadratic polynomial \mathcal{G} over the degree n extension field \mathbb{E} . By construction there is an \mathbb{F} -linear map \mathcal{T}^{-1} such that $\mathcal{T}^{-1} \circ \mathcal{G}$ has rank d , thus there is a rank d matrix that is an \mathbb{E} -linear combination of the Frobenius powers of \mathcal{G} . This turns recovery of the transformation \mathcal{T} into the solution of a MinRank problem over \mathbb{E} .

A significant improvement to this method for HFE is the key recovery attack of Bettale et al. [16]. The first significant observation made was that an \mathbb{E} -linear combination of the *public* polynomials has low rank as a quadratic form over \mathbb{E} . By constructing a formal linear combination of the public polynomials with variable coefficients, one can collect the polynomials representing $(d+1) \times (d+1)$ minors of this linear combination, which must be zero by the Q-rank bound. The advantage this technique offers is that the coefficients of the polynomial are in \mathbb{F} ; thus, the Gröbner basis calculation can be performed over \mathbb{F} , while the variety is computed over \mathbb{E} . This *minors modeling* method is significantly more efficient than the KS-attack when the number of equations is similar to the number of variables. (In contrast, for schemes such as ZHFE, see [17], it seems that the KS modeling is more efficient, probably due to the large number of variables in the Gröbner basis calculation, see [18].) To make the ideal zero-dimensional, we fix one variable; thus, the complexity of the KS-attack with minors modeling is asymptotically $\mathcal{O}(n^{\lceil \log_q(D) \rceil \omega})$, where $2 \leq \omega \leq 3$ is the linear algebra constant.

The MinRank approach can also be effective in attacking HFE-. The key observation in [19] is that not only does the removal of an equation increase the Q-rank by merely one, there is also a basis in which it increases the degree only by a factor of q . Thus HFE- schemes with large base fields are vulnerable to the minors modeling method of [16], even when multiple equations are removed. The complexity of the KS-attack with minors modeling for HFE- is asymptotically

$\mathcal{O}(n^{\lceil \log_q(D) \rceil + a}\omega)$, where a is the number of equations removed and $2 < \omega \leq 3$ is the linear algebra constant.

4 Variants of MinRank with Projection

As first explicitly noted in [15], the Q-rank of the central map is increased by v with the introduction of v vinegar variables and therefore the min-Q-rank of HFEv- is $\lceil \log_q(D) \rceil + a + v$. We now discuss techniques for turning this observation into a key recovery attack. From this point on, let r denote $\lceil \log_q(D) \rceil$, that is, the Q-rank of the HFE component of the central map.

4.1 MinRank then Projection

The simplest way to attempt an attack utilizing the low Q-rank of the central map of HFEv- is to directly apply a MinRank attack and then try to discover the vinegar subspace by considering the solution as a quadratic form. To this end, consider the surjective \mathbb{E} -algebra representation $\Phi : \mathbb{E} \rightarrow \mathbb{A}$ defined by $\Phi(X) = (X, X^q, \dots, X^{q^{n-1}})$. We may map directly from an n -dimensional vector space over \mathbb{F} to \mathbb{A} via right multiplication by the matrix

$$\mathbf{M}_n = \begin{bmatrix} 1 & 1 & \dots & 1 \\ \theta & \theta^q & \dots & \theta^{q^{n-1}} \\ \theta^2 & \theta^{2q} & \dots & \theta^{2q^{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \theta^{n-1} & \theta^{(n-1)q} & \dots & \theta^{(n-1)q^{n-1}} \end{bmatrix},$$

with the choice of a primitive element $\theta \in \mathbb{E}$ (i.e. $\mathbb{E} = \mathbb{F}(\theta)$). Right multiplication by \mathbf{M}_n corresponds to the linear map $\Phi \circ \phi$, where the choice of isomorphism ϕ is determined by the choice of primitive element θ .

We may incorporate the vinegar variables into the picture by simply appending them to \mathbb{A} . Specifically, define the map $\widetilde{\mathbf{M}}_n : \mathbb{F}^{n+v} \rightarrow \mathbb{A} \times \mathbb{F}^v$ by right multiplication by the matrix

$$\widetilde{\mathbf{M}}_n = \begin{bmatrix} \mathbf{M}_n & \mathbf{0}_{n \times v} \\ \mathbf{0}_{v \times n} & I_v \end{bmatrix},$$

where I_v is the identity matrix. We may then represent any HFEv- map as a single $(n + v) \times (n + v)$ matrix with coefficients in \mathbb{E} . Note specifically that any function bilinear with respect to the vinegar variable x_n and the HFE variables x_0, \dots, x_{n-1} can be encoded in row and/or column n of the quadratic form

$$\mathbf{xQx}^\top = \mathbf{x}\widetilde{\mathbf{M}}_n\mathbf{R}\widetilde{\mathbf{M}}_n^\top\mathbf{x}^\top,$$

where $\mathbf{R} \in \mathcal{M}_{(n+v) \times (n+v)}(\mathbb{E})$.

Let \mathbf{F} be defined by $\mathbf{x}\widetilde{\mathbf{M}}_n\mathbf{F}\widetilde{\mathbf{M}}_n^\top\mathbf{x}^\top = \mathcal{F}(x)$ where \mathcal{F} is the central map of HFEv-. We will say that \mathbf{F} is the matrix representation of \mathcal{F} over $\mathbb{A} \times \mathbb{F}^v$. Let

\mathbf{F}^{*i} be the matrix representation of the i th Frobenius power of \mathcal{F} over $\mathbb{A} \times \mathbb{F}_v$. Then we have, for example the following shape for \mathbf{F}^{*0} :

$$\begin{bmatrix} \alpha_{0,0} & \cdots & \alpha_{0,d-1} & 0 \cdots 0 & \beta_{0,n} & \cdots & \beta_{0,n+v-1} \\ \vdots & \ddots & \vdots & \vdots \cdots \vdots & \vdots & \ddots & \vdots \\ \alpha_{0,d-1} & \cdots & \alpha_{d-1,d-1} & 0 \cdots 0 & \beta_{d-1,n} & \cdots & \beta_{d-1,n+v-1} \\ 0 & \cdots & 0 & 0 \cdots 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots \cdots \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 \cdots 0 & 0 & \cdots & 0 \\ \beta_{0,n} & \cdots & \beta_{d-1,n} & 0 \cdots 0 & \beta_{n,n} & \cdots & \beta_{n,n+v-1} \\ \vdots & \ddots & \vdots & \vdots \cdots \vdots & \vdots & \ddots & \vdots \\ \beta_{0,n+v-1} & \cdots & \beta_{d-1,n+v-1} & 0 \cdots 0 & \beta_{n,n+v-1} & \cdots & \beta_{n+v-1,n+v-1} \end{bmatrix}.$$

Here we see that $\text{rank}(\mathbf{F}^{*0}) = r + v$. The structure of \mathbf{F}^{*1} is similar with the upper left HFE block consisting of $\alpha_{i,j}$ shifted down and to the right and raised to the power of q , and the symmetric blocks of mixing monomials shifted down and to the right with a more complicated function applied to the $\beta_{i,j}$ coefficients to respect the Frobenius map.

Now let \mathbf{U} , \mathbf{T} and \mathbf{P}_i be the matrix representations of the affine isomorphisms \mathcal{U} and \mathcal{T} and the public quadratic forms \mathcal{P}_i , respectively. Then we derive the relation

$$(\mathbf{P}_1, \dots, \mathbf{P}_n) \mathbf{T}^{-1} \mathbf{M}_n = (\mathbf{U} \widetilde{\mathbf{M}}_n \mathbf{F}^{*0} \widetilde{\mathbf{M}}_n^\top \mathbf{U}^\top, \dots, \mathbf{U} \widetilde{\mathbf{M}}_n \mathbf{F}^{*(n-1)} \widetilde{\mathbf{M}}_n^\top \mathbf{U}^\top).$$

Thus $\mathbf{U} \widetilde{\mathbf{M}}_n \mathbf{F}^{*0} \widetilde{\mathbf{M}}_n^\top \mathbf{U}^\top$ is an \mathbb{E} -linear combination of the public quadratic forms. Since $\mathbf{U} \widetilde{\mathbf{M}}_n$ is invertible, the rank of this linear combination is the rank of \mathbf{F}^{*0} , which is $r + v$.

Following the analysis of [19, Theorem 2], we see that the effect of the minus modifier on the matrix representation of \mathcal{F} over $\mathbb{A} \times \mathbb{F}^v$ is to add to it constant multiples of itself with a cyclic shift of the rows and columns down and to the right within the HFE block. Thus for HFEEv-, \mathbf{F}^{*0} has the shape given in Fig. 2. The rank of this quadratic form is $r + a + v$.

The solution of the MinRank instance provides an equivalent transformation \mathcal{T}' to the output transformation \mathcal{T} (up to the choice of extension to full rank) and a matrix \mathbf{L} representing the low Q -rank quadratic form $\mathbf{U}' \widetilde{\mathbf{M}}_n \widehat{\mathbf{F}}^{*0} \widetilde{\mathbf{M}}_n^\top \mathbf{U}'^\top$ over $\mathbb{A} \times \mathbb{F}^v$, where $\mathcal{P} = \mathcal{T}' \circ \phi^{-1} \circ \widehat{\mathcal{F}} \circ \phi \circ \mathcal{U}'$ for an equivalent private key $(\mathcal{T}', \widehat{\mathcal{F}}, \mathcal{U}')$. Now that the correct output transformation is recovered, it remains to recover the vinegar subspace of the map \mathcal{L} defined by $\mathbf{L} = \mathbf{U}' \widetilde{\mathbf{M}}_n \widehat{\mathbf{F}}^{*0} \widetilde{\mathbf{M}}_n^\top \mathbf{U}'^\top$.

First, note that the kernel of \mathbf{L} as a linear map is orthogonal to the vinegar subspace, so we may simplify the analysis by projecting onto the orthogonal complement of a codimension one subspace of the kernel. Let $\widehat{\mathcal{L}}$ denote the composition of \mathcal{L} with this projection. The strategy now is to compose codimension one projection mappings π with the transformation $\widehat{\mathcal{L}}$ to filter out the vinegar variables. It suffices to choose projections whose kernels are orthogonal to $\ker(\widehat{\mathbf{L}})$.

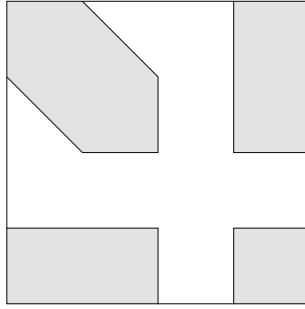


Fig. 2. The shape of the central map of HFEv- composed with the minus projection over $\mathbb{A} \times \mathbb{F}^v$. The shaded areas represent possibly nonzero entries.

If there is a nontrivial intersection between the kernel of π and the vinegar subspace, the rank of the matrix representation of $\widehat{\mathcal{L}} \circ \pi, \mathbf{\Pi} \widehat{\mathbf{L}} \mathbf{\Pi}^\top$, will be reduced. In contrast, if this intersection is empty, the rank of $\mathbf{\Pi} \widehat{\mathbf{L}} \mathbf{\Pi}^\top$ should remain the same. To see this, note that by an argument symmetric to that of [19, Lemma 1] we may equivalently define $\widehat{\mathcal{L}} \circ \pi$ by

$$\widehat{\mathcal{L}} \circ \pi = \mathcal{U}^{-1} \circ [(\phi \circ \pi_1 \circ \phi^{-1} \circ \mathcal{S}_1) \times \pi_2] \circ \mathcal{S}_2,$$

where $\mathcal{S}_1 : \mathbb{F}^n \rightarrow \mathbb{F}^n$ is nonsingular, $\mathcal{S}_2 : \mathbb{F}^{n+v} \rightarrow \mathbb{F}^n \times \mathbb{F}^v$ is an isomorphism, $\pi_1 : \mathbb{E} \rightarrow \mathbb{E}$ has degree at most q^{n-r-a} (since the intersection of the image of $\widehat{\mathcal{L}} \circ \pi$ and the HFE subspace is at least $(r + a)$ -dimensional) and $\pi_2 : \mathbb{F}^v \rightarrow \mathbb{F}^v$ is linear. Since the degree bound of the central HFE quadratic form is q^{r+a} , the highest monomial degree in the composition of π_2 with this map is bounded by q^{n-1} , thus the polynomials $\pi_1, \pi_1^q, \dots, \pi_1^{q^{r+a}}$ are linearly independent.

The probability that the linear form defining $\ker(\pi)$ which is orthogonal to the kernel of $\widehat{\mathbf{L}}$ lies in the vinegar subspace is $q^{-(r+a+1)}$. Once such a vector is recovered, this step is repeated on the orthogonal complement of the discovered vectors until a basis for the vinegar subspace is found. Thus the complexity of this method when fixing one variable to make the ideal zero dimensional is

$$Comp_{MP} = \mathcal{O} \left(\binom{n+r+v}{r+a+v}^2 \binom{n-a}{2} + (r+a+v+1)^3 q^{r+a+1} \right).$$

4.2 Projection then MinRank

Another approach using MinRank is a “project-then-MinRank” approach. In this strategy, one randomly projects the plaintext space onto a codimension k subspace and then applies the MinRank attack. Since the projection π cannot increase the Q-rank of the central map, the Q-rank is at most $r + a + v$.

We may choose $k = n - r - a - v$, and expect that the rank of $\mathcal{P} \circ \pi$ is still $r + a + v$, due to the fact that the HFE component is still of full rank, as noted

in the previous section. If, however, there is a nontrivial intersection between the kernel of π and the vinegar subspace, the rank of this quadratic form will be less than $r + a + v$. The probability this occurs is $q^{k-n} = q^{-(r+a+v)}$.

Generalizing, we may project further in an attempt to eliminate possibly more vinegar variables and reduce the rank further. The minors system of a MinRank attack at rank r is fully determined if the square of r less than the number of variables bounds the number of public equations; thus, if the image of π is of dimension at least the sum of $\sqrt{n-a}$ and r , the minors system is still fully determined. Therefore, consider eliminating c vinegar variables. This requires k to be at least $n - a - r + c - \sqrt{n-a}$. The probability that there is a c -dimensional intersection between the kernel of π and the vinegar subspace is then $q^{c(k-n) - \binom{c}{2}} \geq q^{\binom{c+1}{2} - cr - ca - c\sqrt{n-a}}$.

Once at least one vinegar variable is found, the new basis can be utilized to filter out the remaining vinegar variables as in the previous method. The complexity of the this method with one variable fixed is

$$Comp_{PM} = \mathcal{O} \left(q^{c(r+a+\sqrt{n-a}) - \binom{c+1}{2}} \binom{n+r+v-c}{r+a+v-c}^2 \binom{n-a}{2} \right).$$

5 The Distinguishing Based Attack

In this section we present our distinguishing based attack against the HFEv-signature scheme. We restrict to the case of $\mathbb{F} = \text{GF}(2)$. The idea of the attack is closely related to the direct attacks with projection (also known as the hybrid approach). We define

$$\mathcal{V} = \left\{ \sum_{i=n+1}^{n+v} \lambda_i \mathcal{U}_i \mid \lambda_i \in \{0, 1\} \right\},$$

where \mathcal{U}_i denotes the i -th component of the affine transformation $\mathcal{U} : \mathbb{F}^{n+v} \rightarrow \mathbb{F}^{n+v}$. Therefore, \mathcal{V} is the space spanned by the affine representations of the vinegar variables x_{n+1}, \dots, x_{n+v} . Our attack is based on the following two observations.

- Consider the two HFEv- public keys $\mathcal{P}_1 = \text{HFEv-}(n, D, a, v_1)$ and $\mathcal{P}_2 = \text{HFEv-}(n, D, a, v_2)$. Before applying a Gröbner basis algorithm to the systems, we fix $a + v_1$ variables in \mathcal{P}_1 and $a + v_2$ variables in \mathcal{P}_2 to get determined systems. As shown in Table 1 and Fig. 3, direct attacks against these systems behave differently. In particular, we can distinguish between determined instances of the two systems \mathcal{P}_1 and \mathcal{P}_2 by looking at the step degrees of the F_4 algorithm. This remains possible even when adding (not too many) additional linear equations to the systems \mathcal{P}_1 and \mathcal{P}_2 (thus guessing some of the variables) before applying a Gröbner basis method (hybrid approach).
- Let us consider the special case where $v_2 = v_1 - 1$ holds. By adding one linear equation $\ell \in \mathcal{V}$ to \mathcal{P}_1 , we remove the influence of one of the vinegar

variables from the system \mathcal{P}_1 . A direct attack against the so obtained system \mathcal{P}'_1 therefore behaves in exactly the same way as a direct attack against the system \mathcal{P}_2 (see Table 2).

Table 1. Step degrees of the F_4 algorithm against determined HFEv- systems for different values of v

v	HFEv-(26, 17, 1, v)	HFEv-(33, 9, 3, v)
0	2,3,4,3,4	2,3,4,4,4
1	2,3,4,4,4	2,3,4,5,4
2	2,3,4,5,4	2,3,4,5,5
3	2,3,4,5,5	2,3,4,5,5,5,5,6
4	2,3,4,5,5,5,5,5	2,3,4,5,6,6
5	2,3,4,5,6	
Random system	2,3,4,5,6	2,3,4,5,6,6

5.1 The Distinguisher

Based on the two above observations, we can now construct a distinguisher as follows. We start with an HFEv- public key $\mathcal{P} = \text{HFEv-}(n, D, a, v)$. \mathcal{P} consists of $n - a$ quadratic equations in $n + v$ variables over the field $\text{GF}(2)$. After adding the field equations $\{x_i^2 - x_i : i = 1, \dots, n + v\}$, we append k randomly chosen linear equations ℓ_1, \dots, ℓ_k to the system. Therefore, our new system \mathcal{P}' consists of

- the $n - a$ quadratic HFEv- equations from \mathcal{P}
- $n + v$ field equations $x_i^2 - x_i = 0$ ($i = 1, \dots, n + v$)
- the k linear equations ℓ_1, \dots, ℓ_k .

Altogether, the system \mathcal{P}' consists of $2n - a + v + k$ equations in $n + v$ variables.

After having constructed the system \mathcal{P}' , we solve it via a Gröbner basis algorithm. Due to Observation 2, the behaviour of this algorithm should depend on the fact whether one of the linear equations ℓ_i added to the system (or a linear combination of the ℓ_i) is an element of the vinegar space \mathcal{V} . In fact, we can observe a difference in the step degrees of the algorithm (see Example 1 below).

Formally written, we can use our technique to distinguish between the two cases

$$\left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\} \cap \mathcal{V} = \emptyset \text{ and} \tag{2}$$

$$\left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\} \cap \mathcal{V} \neq \emptyset.$$

However, in most cases that $\left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\} \cap \mathcal{V} \neq \emptyset$, the intersection contains only a single equation $\tilde{\ell}$.

Remark: We have to note here that the number k of linear equations added to the system \mathcal{P} is upper bounded by a value $\bar{k}(n, D, a, v)$. When adding more than \bar{k} linear equations to the system, a distinction between the two cases of (2) is no longer possible.

Example 1: We consider HFEv- systems with $(n, D, a) = (33, 9, 3)$ and varying values of $v \in \{0, \dots, 4\}$. The resulting HFEv- public keys are systems of $n - a = 30$ quadratic equations in $n + v$ variables. After appending the field equations $\{x_i^2 - x_i = 0\}$ to the systems, we added randomly chosen linear equations to reduce the effective number of variables in our systems. Figure 3 shows the degree of regularity of a direct attack using F_4 against the (projected) systems. For comparison, the figure also contains data for a random system of the same size.

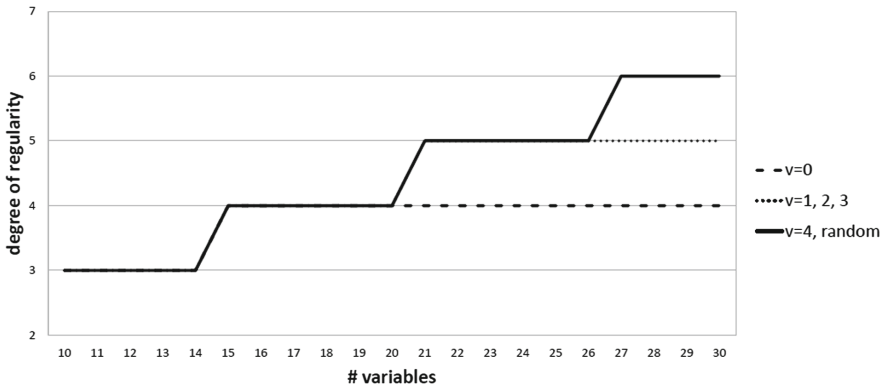


Fig. 3. Direct attack against (projected) HFEv- systems with $(n, D, a) = (33, 9, 3)$ and varying values of v

As Fig. 3 shows, there exists, for every parameter set (n, D, a, v) a number \bar{k} such that

- (1) When adding less than \bar{k} linear equations to the system, the degree of regularity of a direct attack against the projected system is the same as that of a direct attack against the unprojected system.
- (2) When adding $k \geq \bar{k}$ linear equations, the system behaves exactly like a random system of the same size.

Let us now look at our distinguisher. For this, we skip the parameter set $(n, D, a, v) = (33, 9, 3, 0)$ since, in this case, $\mathcal{V} = \emptyset$ holds. However, as Table 2 shows, we can, for each of the values $v \in \{1, \dots, 4\}$, distinguish between the two cases of (2).

For abbreviation, we use in the table $\mathcal{L} := \left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\}$. Note that the evolution of the step degrees for HFEv- $(33, 9, 3, 4)$ is the same as for a random system of the same size.

Table 2. Distinguisher experiments on HFEv-(33, 9, 3, v) systems for different values of v

v	\bar{k}	$n - \bar{k}$	Step degrees of F_4	
			For $\mathcal{L} \cap \mathcal{V} = \emptyset$	For $\mathcal{L} \cap \mathcal{V} = \{\tilde{\ell}\}$
4	3	27	1,2,3,4,5,6	1,2,3,4,5,5,5
3	4	26	1,2,3,4,5,5,5	1,2,3,4,5,5
2	4	26	1,2,3,4,5,5	1,2,3,4,5,4
1	9	21	1,2,3,4,5	1,2,3,4,4,4

5.2 The Attack

Based on the distinguisher presented in the previous section, we can construct an attack against HFEv- as follows. By performing the distinguishing experiment with a large number of systems \mathcal{P}' (containing different linear equations), we can find a set of k linear equations ℓ_1, \dots, ℓ_k such that $\left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\} \cap \mathcal{V} = \{\tilde{\ell}_1\}$. Using this, we can determine the exact form of $\tilde{\ell}_1$ as follows. Note that there exist coefficients $\alpha_i \in \{0, 1\}$ ($i = 1, \dots, k$) such that

$$\tilde{\ell}_1 = \sum_{i=1}^k \alpha_i \cdot \ell_i.$$

In order to determine the exact form of this linear combination, we remove one of the linear equations (say ℓ_1) from the system \mathcal{P}' and add another randomly chosen linear equation. If we still can observe a difference in the behaviour of a direct attack compared to a random choice of linear equations, we know that the coefficient α_1 must be 0. Otherwise, the coefficient α_1 must be 1, and we have to add ℓ_1 back to the system.

We repeat this step for $i = 2, \dots, k$ to determine the values of all the coefficients α_i ($i = 1, \dots, k$). This will give us the exact form of the linear equation $\tilde{\ell}_1 \in \mathcal{V}$. We denote this technique as “remove-and-add” strategy.

Having found $\tilde{\ell}_1$, we add it to the original HFEv-(n, D, a, v) system. The resulting system will behave exactly like an HFEv-($n, D, a, v - 1$) system, and we can again use our distinguisher and repeat the above procedure to find a second linear equation $\tilde{\ell}_2 \in \mathcal{V}$. Note that this will be much easier than finding $\tilde{\ell}_1$ (see next section).

After having found v linear independent equations $\tilde{\ell}_1, \dots, \tilde{\ell}_v \in \mathcal{V}$ and adding them to the HFEv- system, the resulting system will behave exactly like an HFE-(n, D, a) system (i.e. we have no vinegar variables any more). We can then use any attack against HFE- (e.g. the key recovery attack of Vates et al. [19] or a direct attack) to break the scheme. We analyze the complexity of our distinguisher and this attack in the next section.

Let us briefly return to Example 1. When we start with the system $\mathcal{P} = \text{HFEv-}(33, 9, 3, 4)$, we can use our distinguisher to find a set $\{\ell_1, \dots, \ell_k\}$ of linear

equations such that $\left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\} \cap \mathcal{V} = \{ \tilde{\ell}_1 \}$. After having recovered the exact form of $\tilde{\ell}$, we can append it to the system \mathcal{P} , which will then behave exactly like an HFEv-(33, 9, 3, 3) system. Let us denote this new system by $\mathcal{P}^{(1)}$. We can then use the distinguisher on $\mathcal{P}^{(1)}$ to obtain a second linear equation $\tilde{\ell}_2 \in \mathcal{V}$. Adding $\tilde{\ell}_2$ to the system $\mathcal{P}^{(1)}$ leads to a system $\mathcal{P}^{(2)}$ behaving exactly like a HFEv-(33,9,3,2) system. By continuing this process, we finally obtain the system $\mathcal{P}^{(4)}$ corresponding to an HFEv- (33, 9, 3, 0) system. We can then break this scheme by using any attack on HFE-

Algorithm 1. Our distinguishing based attack

Input: HFEv-(n, D, a, v) public key \mathcal{P}

Output: equivalent HFE-(n, D, a) public key $\tilde{\mathcal{P}}$

- 1: Append \tilde{k} randomly chosen linear equations $\ell_1, \dots, \ell_{\tilde{k}}$ in the variables x_1, \dots, x_{n+v} (as well as the field equations $x_i^2 - x_i = 0$) to the system \mathcal{P} and solve it by F_4 .
 - 2: Repeat this step until the F_4 -step degrees differ from the standard case. This means that we have found a set of linear equations ℓ_1, \dots, ℓ_k such that $\left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\} \cap \mathcal{V} = \{ \tilde{\ell}_1 \}$
 - 3: Determine the exact form of $\tilde{\ell}$ by the above described “remove-and-add” strategy.
 - 4: Append the linear equation $\tilde{\ell}$ to the system \mathcal{P} . The resulting system \mathcal{P}' will behave exactly like an HFEv-($n, D, a, v-1$) public key.
 - 5: Repeat the above steps until having found v linear independent equations $\tilde{\ell}_1, \dots, \tilde{\ell}_k \in \mathcal{V}$.
 - 6: **return** $\tilde{\mathcal{P}} = (\mathcal{P}, \tilde{\ell}_1, \dots, \tilde{\ell}_v)$
-

6 Complexity Analysis

In the first step of our attack, we have to find one linear equation $\tilde{\ell} \in \mathcal{V}$ by using our distinguisher and a following application of the “remove-and-add” strategy described in the previous section. Therefore, the complexity of this first step of our attack is determined by three factors:

1. The number of times we have to run the distinguisher in order to find a set of linear equations ℓ_1, \dots, ℓ_k such that $\left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\} \cap \mathcal{V} = \{ \tilde{\ell} \}$,
2. The cost of one run of the distinguisher and
3. The cost of recovering the exact form of $\tilde{\ell}$.

The first number is determined by

- The probability that a randomly chosen linear equation in $n + v$ variables is contained in the space \mathcal{V} spanned by the linear representation of the vinegar variables $\mathcal{U}_{n+1}, \dots, \mathcal{U}_{n+v}$. A randomly chosen linear equation $\tilde{\ell}$ in $n + v$ variables can be seen as a linear combination of the components of \mathcal{U} , i.e.

$$\tilde{\ell} = \sum_{i=1}^{n+v} \lambda_i \cdot \mathcal{U}_i. \tag{3}$$

The reason for this is that \mathcal{U} is an invertible map from \mathbb{F}^{n+v} to itself, which means that the components of \mathcal{U} form a basis of this space. There are 2^{n+v} choices for the parameters λ_i ($i = 1, \dots, n + v$). On the other hand, every element $\tilde{\ell}$ of the space \mathcal{V} spanned by the linear transformations of the vinegar variables v_1, \dots, v_v can be written in the form

$$\tilde{\ell} = \sum_{i=n+1}^{n+v} \lambda_i \cdot \mathcal{U}_i.$$

The probability that a randomly chosen linear equation $\bar{\ell}$ lies in \mathcal{V} is therefore given by

$$\text{prob}(\bar{\ell} \in \mathcal{V}) = 2^{-n}. \tag{4}$$

The reason for this is that all the coefficients λ_i ($i = 1, \dots, n$) in the representation (3) of $\bar{\ell}$ must be zero.

- The number of linear equations (and linear combinations thereof) added to the public key. When adding k linear equations ℓ_1, \dots, ℓ_k to the public key, we do not have to consider only the k equations ℓ_1, \dots, ℓ_k itself, but also all linear combinations of the form

$$\ell = \sum_{i=1}^k \lambda_i \cdot \ell_i.$$

The total number of linear equations we have to consider is therefore not k , but 2^k .

Therefore, when adding k linear equations ℓ_1, \dots, ℓ_k to the public key, the probability of finding one linear equation $\tilde{\ell} \in \mathcal{V}$, is given by

$$\text{prob} = 1 - (1 - 2^{-n})^{2^k} \approx 2^{k-n}.$$

In order to find one linear equation $\tilde{\ell} \in \mathcal{V}$, we therefore have to run our distinguisher about 2^{n-k} times.

A single run of our distinguisher corresponds to one run of the F_4 algorithm. The cost of this can be estimated as

$$\text{Comp}_{F_4} = 3 \cdot \binom{n'}{d_{\text{reg}}}^2 \cdot \binom{n'}{2},$$

where n' is the number of variables in the quadratic system and d_{reg} is the so called degree of regularity.

Note that this formula assumes that the linear systems appearing during the attack are solved using a sparse Wiedemann solver. Furthermore we use the fact that the system is defined over the field $\text{GF}(2)$, which reduces the number of terms in the high degree polynomials.

With regard to the number n' of variables we find that the linear equations added to the public key are “absorbed” at a very early step of the F_4 algorithm,

i.e. they are used to reduce the number of variables in the system. This fact is illustrated in Table 3. In the table, we consider two random systems, both containing 25 quadratic equations. However, while the equations of system A are polynomials in 25 variables, the polynomials of system B contain 35 variables. On the other hand, the system B additionally contains 10 linear equations.

Table 3. Experiments with random systems

Step	25 equations, 25 variables			25 quadr. + 10 lin. equations, 35 variables		
	Degree	Matrix size	Time	Degree	Matrix size	Time
				1	10 × 36	0.0
				1	20 × 36	0.0
1	2	25 × 326	0.0	2	330 × 631	0.0
2	3	652 × 2626	0.02	3	650 × 2626	0.02
3	4	7,894 × 14,498	1.27	4	7864 × 15568	1.34
4	5	52,488 × 52,956	79.86	5	52197 × 52665	80.26
5	6	248,705 × 245,506	179.34	6	248,273 × 108,524	182.24

As the table shows, both systems behave very similarly. Starting at step 2 (degree 3), there is no significant difference between the matrix sizes or the running times of the single steps between the two systems.

We can therefore conclude that the quadratic systems we consider in our distinguishing based attack ($n - a$ quadratic equations + k linear equations in $n + v$ variables) behave just like systems of $n - a$ quadratic equations in $n + v - k$ variables.

The cost of recovering the exact form of $\tilde{\ell}$ is negligible in comparison to finding linear equations ℓ_1, \dots, ℓ_k such that $\left\{ \sum_{i=1}^k \lambda_i \ell_i \mid \lambda_i \in \{0, 1\} \right\} \cap \mathcal{V} = \{\tilde{\ell}\}$. Remember that $\tilde{\ell}$ can be written as a linear combination of ℓ_1, \dots, ℓ_k , i.e. $\tilde{\ell} = \sum_{i=1}^k \lambda_i \cdot \ell_i$.

As described in the previous section, we remove for this one linear equation ℓ_i from the system \mathcal{P}' . By adding a randomly chosen linear equation, we obtain a system \mathcal{P}'' of the same dimensions. We apply the F_4 algorithm against the two systems \mathcal{P}' and \mathcal{P}'' . If we observe a difference in the behavior of the algorithm, we know that the coefficient λ_i in the above linear combination is 1. Otherwise we have $\lambda_i = 0$. By running this test for all $i \in \{1, \dots, k\}$, we can determine all the coefficients λ_i and therefore recover $\tilde{\ell}$. In order to recover $\tilde{\ell}$, we therefore need $2 \cdot k$ runs of the F_4 algorithm, which is far less than the 2^{n-k} F_4 -runs above. Therefore, we do not have to consider this step in our complexity analysis.

Altogether, we can estimate the complexity of this first step of our attack by

$$Comp_{Dist; \text{classical}} = 2^{n-k} \cdot 3 \cdot \binom{n+v-k}{d_{\text{reg}}}^2 \cdot \binom{n+v-k}{2}. \quad (5)$$

In the presence of quantum computers, we can speed up the searching step of this attack using Grover’s algorithm. Thus we get

$$Comp_{Dist; \text{ quantum}} = 2^{(n-k)/2} \cdot 3 \cdot \binom{n+v-k}{d_{\text{reg}}}^2 \cdot \binom{n+v-k}{2}.$$

Note that this assumption of the complexity is very optimistic, since it assumes a perfect “square-root” speed up by Grover’s algorithm. Since quantum algorithms must be reversible, it is not clear if this is possible.

As Eq. (5) shows, the complexity decreases when we increase the number k of linear equations added to the public key. However, as already mentioned in the previous section, our distinguisher fails when k is too large. We denote the maximal value of k for which our distinguisher works by $\bar{k}(n, D, a, v)$.

In order to remove all the vinegar variables from the system \mathcal{P} , we have to repeat the above process v times. However, with decreasing v we find (see Table 2)

- (1) the number \bar{k} of linear equations that we can add to the public system increases, reducing the number of F_4 -runs.
- (2) the degree of regularity of the systems generated by our distinguisher decreases, reducing the complexity of a single F_4 -run.

Therefore, the following steps of our attack will be much faster than the first step. This means, that we can estimated the complexity of the whole attack as in formula (5).

However, in order to estimate the complexity of our attack against an HFEv- (n, D, a, v) scheme in practice, we still have to answer the following two questions.

- What is the maximal number \bar{k} of linear equations we can add to the public key such that our distinguisher works?
- What is the degree of regularity of the systems generated by our distinguisher?

In order to answer these questions, we once more consider Example 1 (see previous section).

First, let us consider the second question. As a comparison of Table 2 and Fig. 3 shows, the degree of regularity of solving the systems generated by our distinguisher corresponds exactly to the degree of regularity of an unprojected HFEv- system with parameters (n, D, a, v) . As stated in [20], we can estimate this value as

$$d_{\text{reg}} = \left\lceil \frac{r + a + v + 7}{3} \right\rceil, \tag{6}$$

where $r = \lceil \log_q(D - 1) \rceil + 1$.

To answer the first question, let us take a closer look on the behavior of the hybrid approach against random systems (see Fig. 3). We start with a random system of 30 quadratic equations in 30 variables over GF(2). After appending the field equations $x_i^2 - x_i = 0$ ($i = 1, \dots, 30$), we add $k \in \{0, \dots, 20\}$ linear equations to the system. Table 4 shows for which values of k we reach given values of regularity.

Table 4. Degree of regularity of projected random systems with 30 equations

d_{reg}	# k of added linear equations
3	For $k \geq 16$
4	For $10 \leq k \leq 15$
5	For $4 \leq k \leq 9$
6	For $k \leq 3$

Let us define $\hat{k}(d)$ to be the maximal number of linear equations we can add to the random system, such that the degree of regularity of a direct attack against the system is greater or equal to d , i.e $\hat{k}(6) = 3$, $\hat{k}(5) = 9$ and $\hat{k}(4) = 15$.

By comparing these numbers with the values of \bar{k} listed in Table 2, we find

$$\hat{k}(d^*) \leq \bar{k} \leq \hat{k}(d^*) + 1,$$

where d^* is the degree of regularity of a direct attack against an HFEv- (n, D, a, v) scheme (see Eq. (6)).

In order to estimate the complexity of our attack against an HFEv- (n, D, a, v) scheme, we therefore proceed as follows.

1. We compute the degree of regularity of the unprojected HFEv- (n, D, a, v) system (see Eq. (6)). Denote the result by d^* .
2. We estimate the maximal number \bar{k} of linear equations we can add to the public HFEv- system by $\hat{k}(d^*)$. This value can be obtained as follows.

The degree of regularity of a random system of $m = n - a$ quadratic equations in n' variables over $\text{GF}(2)$ can be estimated as the smallest index d for which the coefficient of X^d in

$$\frac{1}{1 - X} \cdot \left(\frac{1 - X^2}{1 - X} \right)^{n'} \cdot \left(\frac{1 - X^2}{1 - X^4} \right)^m$$

is non-positive [21].

We can use this equation to determine the values of $\hat{k}(d^*)$.

By substituting the so obtained values of \bar{k} and d^* into formula (5), we therefore get a close estimation of the complexity of our distinguishing based attack against an HFEv- (n, D, a, v) system.

Remark: The above procedure allows to get an estimation of the complexity of our distinguishing based attack against a given HFEv- scheme. However, it seems to be a very hard task to find a closed formula for this complexity.

Example 2: Consider an HFEv- system over $\text{GF}(2)$ with $(n, D, a, v) = (91, 5, 3, 2)$. We obtain $r = \lfloor \log_2(D - 1) \rfloor + 1 = 3$. The degree of regularity of a direct attack against the HFEv- system (with field equations) is given by

$$d_{\text{reg}} = \left\lfloor \frac{3 + 3 + 2 + 7}{3} \right\rfloor = 5.$$

Therefore, we get

$$Comp_{\text{direct}} = 3 \cdot \binom{88}{5}^2 \cdot \binom{88}{2} \approx 2^{63.9}.$$

After adding $k = 68$ randomly chosen linear equations to the system, the step degrees of the $F4$ algorithm look like 1; 1, 2, 3, 4. When one of the linear equation was chosen from the vinegar space \mathcal{V} , we obtain 1; 1, 2, 3, 3.

Therefore, we can estimate the complexity of our distinguisher by

$$Comp_{\text{Distinguisher}} = 2^{23} \cdot \binom{25}{4}^2 \cdot \binom{25}{2} \approx 2^{60.1},$$

which is nearly 16 times faster than a direct attack.

The “MinRank-then-project” approach has a complexity estimated by

$$Comp_{\text{MP}} = 3 \cdot \binom{96}{8}^2 \binom{88}{2} \approx 2^{87.4},$$

while the complexity of the “project-then-MinRank” approach has complexity

$$Comp_{\text{PM}} = 2^{14} \cdot 3 \cdot \binom{95}{7}^2 \binom{88}{2} \approx 2^{92.6}.$$

Therefore, for the above parameter set, the distinguishing based attack is the most efficient classical attack against HFEv-.

With regard to the memory consumption, we get

$$\text{Memory}_{\text{direct}} = \binom{88}{5}^2 \approx 2^{50.4},$$

$$\text{Memory}_{\text{MP}} = \binom{96}{8}^2 \approx 2^{73.9},$$

$$\text{Memory}_{\text{PM}} = \binom{95}{7}^2 \approx 2^{66.7},$$

$$\text{Memory}_{\text{Distinguisher}} = \binom{25}{4}^2 \approx 2^{27.3}.$$

As these data show, the distinguishing based attack requires much less memory than the direct and the MinRank attack. Since attacks against large instances of multivariate schemes often fail due to memory restrictions, the small memory consumption is a huge benefit of this attack.

Remark: The comparably low complexity of our attack in Example 2 is caused by the small number of vinegar variables in the system. Due to this, the distinguisher works also for relatively small numbers of variables, which enables us to add a large number of linear equations to the system. This again reduces the number of distinguisher runs and therefore the complexity of the attack. (In the case of the example, we found that the distinguisher works for only 25 variables in the system, due to which we had to run our distinguisher only 2^{23} times.)

When the number v of vinegar variables increases, we can not distinguish between the two cases at 25 variables any more. We have to reduce the number of linear equations added to the system and therefore have to run the distinguisher much more often (and for larger systems). Therefore, for larger values of v , the complexity of our attack increases.

For the parameter sets usually used in HFEEv- like schemes (and suggested for the NIST call for proposals), the direct attack is usually more efficient than our attack. However, in terms of memory consumption, our attack is still much better.

7 Possible Future Work

In this section we shortly describe a strategy to reduce the complexity of our attack. However, since we have neither enough space nor time to present our idea completely, we leave a detailed analysis as future work.

In the distinguishing step of our attack, we solve a large number of multivariate systems using a direct attack. These systems are obtained by adding k linear equations to a multivariate quadratic system \mathcal{P} of m equations in $n + v$ variables (or equivalently projecting the system to a $n + v - k$ dimensional subspace). In Sect. 5, these projections were chosen at random.

The main idea to reduce the complexity of this step is now to select the projection in a slightly nonrandom fashion. In particular, we consider a projection in two steps. We apply a projection $\tilde{\pi}$ of corank $k + 1$ to the system \mathcal{P} and derive from this a set of corank k projections $\{\pi_i\}$. In this case, we can treat the image of $\tilde{\pi}$ in the plaintext space as being generated by the variables $x_1, \dots, x_{n+v-k-1}$, while the image of each of the projections π_i is generated by the same variables plus one additional variable x_{n+v-k} , which defines a 1-dimensional subspace of the kernel of $\tilde{\pi}$, which will vary depending on the choice of π_i .

During the computation of a Gröbner basis of $\mathcal{P}(\pi_i) = (f_1(\pi_i), \dots, f_m(\pi_i))$, the F_4 algorithm looks for polynomials p_j of degree $d-2$ such that $\sum p_j \cdot f_j(\pi_i) = q$, where q is a polynomial of degree at most $d - 1$.

Our strategy will be to first solve for all p_j in the variables $x_1, \dots, x_{n+v-k-1}$, such that

$$\sum p_j f_j(\pi_i) = q \pmod{x_{n+v-k}}.$$

As the above equation is equivalent to $\sum p_j f_j(\tilde{\pi}) = q$, this computation can be reused for multiple different choices of π_i . By doing so, we therefore can reduce the effort of computing the Gröbner basis needed during the application of our distinguisher.

However, in order to find the exact amount of saving, much more work is required. We therefore leave an exact analysis of the above mentioned idea as future work.

Another topic for future work is a precise complexity analysis of our attack. The complexity analysis presented in Sect. 6 is based much on heuristics and experiments. In particular, formula (5) contains the parameters \bar{k} and d_{reg}^* , which

(so far) could only be determined experimentally. It therefore would be desirable to develop a formula which computes the complexity of our attack for given HFE v - parameters n , D , a and v .

Acknowledgements. We thank the anonymous reviewers of PQCrypto for their valuable comments which helped to improve this paper. In particular we want to thank the shepherd for her help in creating the final version of this paper.

References

1. 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
2. Ding, J., Schmidt, D.: Rainbow, a new multivariable polynomial signature scheme. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 164–175. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_12
3. Patarin, J., Courtois, N., Goubin, L.: QUARTZ, 128-bit long digital signatures. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 282–297. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45353-9_21
4. Bettale, L., Faugère, J.C., Perret, L.: Hybrid approach for solving multivariate systems over finite fields. *J. Math. Cryptol.* **3**, 177–197 (2009)
5. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York (1979)
6. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_4
7. 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
8. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE public key cryptosystem by relinearization. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_2
9. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_27
10. Faugere, J.C.: A new efficient algorithm for computing Gröbner bases (F4). *J. Pure Appl. Algebra* **139**, 61–88 (1999)
11. Faugere, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (f5). In: ISSAC 2002, pp. 75–83. ACM Press (2002)
12. Mohamed, M.S.E., Ding, J., Buchmann, J.: Towards algebraic cryptanalysis of HFE challenge 2. In: Kim, T., Adeli, H., Robles, R.J., Balitanas, M. (eds.) ISA 2011. CCIS, vol. 200, pp. 123–131. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23141-4_12
13. Ding, J., Hodges, T.J.: Inverting HFE systems is quasi-polynomial for all fields. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 724–742. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_41

14. Ding, J., Kleinjung, T.: Degree of regularity for HFE-. IACR Cryptology ePrint Archive **2011**, 570 (2011)
15. Ding, J., Yang, B.-Y.: Degree of regularity for HFEv and HFEv-. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 52–66. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_4
16. Bettale, L., Faugère, J., Perret, L.: Cryptanalysis of HFE, multi-HFE and variants for odd and even characteristic. Des. Codes Cryptogr. **69**, 1–52 (2013)
17. Porras, J., Baena, J., Ding, J.: ZHFE, a new multivariate public key encryption scheme. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 229–245. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11659-4_14
18. Cabarcas, D., Smith-Tone, D., Verbel, J.A.: Key recovery attack for ZHFE. [22], pp. 289–308 (2017). https://doi.org/10.1007/978-3-319-59879-6_17
19. Vates, J., Smith-Tone, D.: Key recovery attack for all parameters of HFE-. [22], pp. 272–288 (2017). https://doi.org/10.1007/978-3-319-59879-6_16
20. Petzoldt, A.: On the complexity of the hybrid approach on HFEv-. Cryptology ePrint Archive, Report 2017/1135 (2017). <https://eprint.iacr.org/2017/1135>
21. Yang, B.-Y., Chen, J.-M.: Theoretical analysis of XL over small fields. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 277–288. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27800-9_24
22. Lange, T., Takagi, T. (eds.): PQCrypto 2017. LNCS, vol. 10346. Springer, Cham (2017). <https://doi.org/10.1007/978-3-319-59879-6>



HFERP - A New Multivariate Encryption Scheme

Yasuhiko Ikematsu³, Ray Perlner², Daniel Smith-Tone^{1,2}, Tsuyoshi Takagi³,
and Jeremy Vates¹(✉)

¹ Department of Mathematics, University of Louisville, Louisville, KY, USA
jeremy.vates@louisville.edu

² National Institute of Standards and Technology, Gaithersburg, MD, USA
{ray.perlner,daniel.smith}@nist.gov

³ Institute of Mathematics for Industry, Kyushu University, Fukuoka, Japan
{y-ikematsu,takagi}@imi.kyushu-u.ac.jp

Abstract. In 2016, Yasuda et al. presented a new multivariate encryption technique based on the Square and Rainbow primitives and utilizing the plus modifier that they called SRP. The scheme achieved a smaller blow-up factor between the plaintext space and ciphertext space than most recent multivariate encryption proposals, but proved to be too aggressive and was completely broken by Perlner et al. in 2017. The scheme suffered from the same MinRank weakness that has allowed effective attacks on several notable big field multivariate schemes: HFE, multi-HFE, HFE-, for example.

We propose a related new encryption scheme retaining the desirable traits of SRP and patching its weaknesses. We call the scheme HFERP because it utilizes a similar construction as SRP with an HFE primitive replacing the Square polynomial. The effect of this substitution is to increase the Q-rank of the public key to such a degree that the MinRank attack is impossible. HFERP still retains the relatively small blow-up factor between the plaintext space and ciphertext space, and is thus a candidate for secure multivariate encryption without an essential doubling in size between plaintext and ciphertext.

Keywords: Multivariate cryptography · HFE · Encryption
MinRank · Q-rank

1 Introduction

Ever since the discovery of polynomial time algorithms for factoring and computing discrete logarithms on a quantum computer by Shor [1], creating schemes that resist such developments has fallen upon the shoulders of today's cryptographers. In recent years, quantum computing has made significant advances

The rights of this work are transferred to the extent transferable according to title 17 § 105 U.S.C.

leading some experts to make more confident predictions that the post-quantum world will soon be upon us, see, for example, [2].

There has also been an explosive development in public key technologies relying on mathematics for which there is no known significant computational advantage quantum computers possess. In particular, multivariate public key cryptography (MPKC) produced numerous schemes for public key encryption and digital signatures in the late 1990s. These schemes further fueled the development of computational algebraic geometry, and seem to have inspired the advancement of some of the symbolic algebra techniques we now apply to all areas of post-quantum cryptography, that is, cryptography designed with quantum computers in mind.

With the development of such techniques, many multivariate schemes have been cryptanalyzed and broken. Specifically, multivariate encryption seems to be challenging. The purpose of this article is to confront this challenge, advancing a new multivariate encryption scheme HFERP, based on SRP, see [3], developed to eradicate the deficiencies of its predecessor.

1.1 Recent History

While there may be many trustworthy candidates for multivariate signatures, such as UOV [4], Rainbow [5], and Gui [6], developing multivariate schemes for encryption has been a bit of a struggle. While some older ideas have been reborn with better parameter sets due to the advancement of the science, such as applying HFE-, see [7], to encryption, most of the surviving multivariate encryption schemes are relatively young.

In the last few years, there have been a few new proposals for multivariate encryption, mostly following the idea that it is easier to hide the structure of an injective mapping into a large codomain than to hide the structure of a bijection, as is needed for any encryption mapping into a codomain of the same size as the domain. The ABC Simple Matrix encryption scheme of [8, 9] and ZHFE, see [10] are examples of this idea. Most of these encryption ideas, both new and old, have inspired recent surprising cryptanalyses that affect parameter selection or outright break the scheme, see [11–15], for example.

Such a tale describes the life of SRP, see [3], the design of which aimed to be very efficient and holds a comparably small blow up factor between the plaintext and ciphertext sizes. The scheme also claimed security against attacks efficient against the Square and Rainbow schemes by combining them into one. Unfortunately, SRP is also the victim of a new cryptanalysis, see [16]. The attack exploits the low Q -rank of the Square map, a vulnerability inherited by the public key. A modified MinRank attack was able to pull apart the Square polynomials from the Rainbow and Plus polynomials in the public key.

1.2 Our Contribution

We present a new composite scheme in the manner of SRP by replacing the weaker Square layer with an HFE polynomial of higher Q -rank and finding the

correct balance in the sizes of the HFE, Rainbow and Plus layers for efficiency and security. We call our scheme HFERP. We further establish the complexity of the relevant attack models: the algebraic attack, the MinRank attack, and the invariant attack.

1.3 Organization

The paper is organized as follows. In the next section, we present isomorphisms of polynomials and describe the structure of HFE and SRP. The subsequent section reviews the Q-rank of ideals in polynomial rings and discusses invariant properties of Q-rank and min-Q-rank. In Sect. 4, we review more carefully the previous cryptanalyses of HFE and SRP. We then present HFERP in the next section. Section 6 discusses the complexity of all known relevant attacks on HFERP. Our choice of parameters to optimize security and performance along with experimental results are then presented in the following section. Finally, we conclude discussing why a similar approach to SRP seems to produce such a different technology in HFERP.

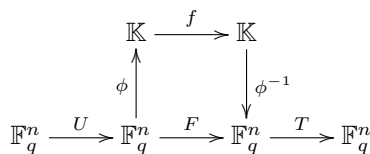
2 Big Field Schemes

HFE and SRP are members of a family of cryptosystems known as “big field” schemes. This term is based on the system exploiting the vector space structure of a degree n extension of \mathbb{K} over a finite field \mathbb{F}_q . Using core maps within the extension field allows us to take advantage of Frobenius automorphisms $x \mapsto x^q$ for any function of the form $f(x) = x^{q^i + q^j}$, noting that $\phi^{-1} \circ f \circ \phi$ is a vector-valued quadratic function over \mathbb{F}_q where $\phi : \mathbb{F}_q^n \rightarrow \mathbb{K}$ is an \mathbb{F}_q -vector space isomorphism. By observing that any vector-valued quadratic function on \mathbb{F}_q^n is isomorphic to a sum of such monomials, it is clear that any quadratic function f over \mathbb{K} can be represented as a vector-valued function, F , over \mathbb{F}_q .

This equivalence allows us to construct cryptosystems in conjunction with the following concept, the isomorphisms of polynomials.

Definition 1. *Two vector-valued multivariate polynomials F and G are said to be isomorphic if there exist two affine maps T, U such that $G = T \circ F \circ U$.*

The equivalence and isomorphism marry in a method commonly referred to as the butterfly construction. Given a vector space isomorphism $\phi : \mathbb{F}_q^n \rightarrow \mathbb{K}$ and an efficiently invertible map $f : \mathbb{K} \rightarrow \mathbb{K}$, we compose two affine transformations $T, U : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^n$ in order to obscure our choice of basis for the input and output. This construction generates a vector-valued map $P = T \circ \phi^{-1} \circ f \circ \phi \circ U = T \circ F \circ U$, where $F = \phi^{-1} \circ f \circ \phi$.



2.1 HFE

The Hidden Field Equation Scheme was first introduced by Patarin, see [7], as an improvement on the well known C^* construction of [17]. Patarin's contribution was to use a general polynomial with degree bound D in place of the central monomial map of C^* .

Explicitly, one chooses a quadratic map $f : \mathbb{K} \rightarrow \mathbb{K}$ of the form:

$$f(x) = \sum_{\substack{i \leq j \\ q^i + q^j \leq D}} \alpha_{i,j} x^{q^i + q^j} + \sum_{\substack{i \\ q^i \leq D}} \beta_i x^{q^i} + \gamma, \quad (1)$$

where the coefficients $\alpha_{i,j}, \beta_i, \gamma \in \mathbb{K}$ and the degree bound D is sufficiently low for efficient inversion using the Berlekamp algorithm, see [18].

The public key is computed as $P = T \circ F \circ U$, where $F = \phi^{-1} \circ f \circ \phi$. Inversion is accomplished by taking a ciphertext $y = P(x)$, computing $v = T^{-1}(y)$, solving $\phi(v) = f(u)$ for u via the Berlekamp algorithm and then recovering $x = U^{-1}(\phi^{-1}(u))$.

2.2 Rainbow

The Rainbow scheme is a generalization of Patarin's UOV, see [4]. The key idea, introduced by Ding, see [5], was constructing multiple layers of UOV.

Let \mathbb{F} be a finite field with a degree n extension \mathbb{F}^n . Let $\mathcal{V} = \{1, 2, \dots, n\}$. For a chosen u , let v_1, \dots, v_u be integers such that $0 < v_1 < \dots < v_u = n$ and let $\mathcal{V}_l = \{1, \dots, v_l\}$ for each $l \in \{1, \dots, u\}$. Note that $|\mathcal{V}_i| = v_i$.

Let $o_i = v_{i+1} - v_i$ for each $i \in \{1, \dots, u-1\}$ and $\mathcal{O}_i = S_{i+1} - S_i$ for each $i \in \{1, \dots, u-1\}$. Define P_l to be the space generated by the span of polynomials of the following form:

$$f(x_1, \dots, x_n) = \sum_{i \in \mathcal{O}_l, j \in \mathcal{V}_l} \alpha_{i,j} x_i x_j + \sum_{i,j \in \mathcal{V}_l} \beta_{i,j} x_i x_j + \sum_{i \in \mathcal{V}_l} \gamma_i x_i + \eta$$

One can refer to the previous constructions using the following terminology: \mathcal{O} is the collection of oil variables, \mathcal{V} is the collection of vinegar variables, and a polynomial $f \in P_l$ is an l -th layer Oil and Vinegar polynomial.

The Rainbow map $F : \mathbb{F}^n \rightarrow \mathbb{F}^{n-v_1}$ is defined as (with x_1, \dots, x_n being referred to as \bar{x} for convenience)

$$F(\bar{x}) = (\tilde{F}_1(\bar{x}), \dots, \tilde{F}_{u-1}(\bar{x})) = (F_1(\bar{x}), \dots, F_{n-v_1}(\bar{x}))$$

where each \tilde{F}_i consists of o_i randomly chosen quadratic polynomials from P_i . F is a Rainbow polynomial map with $u-1$ layers. The public key is generated in the usual fashion by applying two affine transformations, T and U , where $T : \mathbb{F}^{n-v_1} \rightarrow \mathbb{F}^{n-v_1}$ and $U : \mathbb{F}^n \rightarrow \mathbb{F}^n : T \circ F \circ U$.

2.3 SRP

In Sect. 5, we present in detail the construction of our proposed scheme, HFERP. For reference, we will include the Square Map definition as well as method of inversion presented in the original SRP paper, see [3].

Instead of using the HFE core map described in Sect. 5, SRP uses the Squaring map where the Square component is defined as $\mathcal{F}_S : \mathbb{F}_q^{n'} \rightarrow \mathbb{F}_q^d$ (where $q^d + 1$ is divisible by 4) and it is the result of the following composition:

$$\mathbb{F}_q^{n'} \xrightarrow{\pi_d} \mathbb{F}_q^d \xrightarrow{\phi} \mathbb{K} \xrightarrow{X \mapsto X^2} \mathbb{K} \xrightarrow{\phi^{-1}} \mathbb{F}_q^d$$

Upon inversion step 3, the user would compute

$$R_{1,2} = \pm X^{(q^d+1)/4}$$

and use it to find $\mathbf{y} = (y_1^{(i)}, \dots, y_d^{(i)}) = \phi^{-1}(R_i) \in \mathbb{F}_q^d$. The choice of the Square map was made because of the speed of inversion it provided when compared to any other quadratic maps. Unfortunately, due to this choice, SRP was quickly broken in [16] by isolating the squaring public polynomials and exploiting its low Q-rank.

3 Q-Rank

The min-Q-rank of the public key is a critical quantity when analyzing the security of big field schemes within multivariate cryptography. For clarification, the definition is as follows:

Definition 2. *The Q-rank of any quadratic map $f(\bar{x})$ on \mathbb{F}_q^n is the rank of the quadratic form $\phi^{-1} \circ f \circ \phi$ in $\mathbb{K}[X_0, \dots, X_{n-1}]$ via the identification $X_i = \phi(\bar{x})^{q^i}$.*

Usually, the definition of the rank of a quadratic form is given as the minimum number of variables required to express an equivalent quadratic form due to quadratic form equivalences corresponding to matrix congruence. Note that congruent matrices have the same rank. This same quantity is equal to the rank of the matrix representations of the quadratic form, even in characteristic 2, where the quadratics x^{2q^i} are additive, but not linear for $q > 2$.

Q-rank is invariant under one-sided isomorphisms $f \mapsto f \circ U$, but is not invariant under isomorphisms of polynomials in general. The quantity that is often meant by the term Q-rank, but more properly called min-Q-rank, is the minimum Q-rank among all nonzero linear images of f . This min-Q-rank is invariant under isomorphisms of polynomials and is the quantity relevant for cryptanalysis.

4 Previous Cryptanalysis of Relevant Schemes

SRP was designed as a concatenation of two known multivariate schemes and a scheme modifier. The first component was Square, see [19], which can be seen as a degenerate version of HFE. The second component was oil-and-vinegar (OV) or, more generally, Rainbow, see [5, 20]. The final component was the plus modifier, first proposed in [21]. The algebraic properties of these schemes were intended to complement their weaknesses when used in conjunction. This patchwork design requires, however, a careful consideration of the relevant cryptanalyses within all of these families.

The original oil-and-vinegar (OV) scheme, proposed in [20], was completely broken in [22] by what we call the invariant method. Specifically, the balanced OV scheme contains an equal number of oil variables, variables which only occur linearly in the central map, and vinegar variables, which occur quadratically. Thus, the differential of any central polynomial has the shape

$$Df_i = \begin{bmatrix} a_{1,1} & \cdots & a_{1,v} & a_{1,v+1} & \cdots & a_{1,2v} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{1,v} & \cdots & a_{v,v} & a_{v,v+1} & \cdots & a_{v,2v} \\ a_{1,v+1} & \cdots & a_{v,v+1} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{1,2v} & \cdots & a_{v,2v} & 0 & \cdots & 0 \end{bmatrix},$$

under an appropriate basis of $\mathbb{F}^{2v} = V \oplus O$, where V is the subspace spanned by the vinegar variables and O is the subspace spanned by the oil variables.

The invariant attack proceeds by computing the differential of random linear combinations of the public polynomials until two full rank differentials, Df_1 and Df_2 , are produced. Then O is left invariant by $Df_1^{-1}Df_2$ and is thus easily recovered. A similar technique has been used in conjunction with rank attacks to assault schemes with a similar structure whenever $\dim(V) \leq \dim(O)$, see, in particular, [11–13].

HFE and some of its modifications have been the target of effective cryptanalyses utilizing the low Q -rank property of the central map. Each of these cryptanalyses can be described as a big field MinRank attack, recovering a low rank quadratic form over the extension \mathbb{E} from which an isomorphism relating the public key to an equivalent private key can be derived.

The earliest iteration of this technique is the well-known Kipnis-Shamir (KS) attack of [23], also known by the name MinRank, due to the close relationship between the attack and the MinRank problem in algebraic complexity theory, see [24]. The KS-attack recovers a private key for HFE by exploiting the fact that the low Q -rank of the central map is a property preserved by isomorphisms. Considering an odd characteristic instance of HFE. We may write the homogeneous quadratic part of the central map as

$$\begin{bmatrix} x & x^q & \cdots & x^{q^{n-1}} \end{bmatrix} \begin{bmatrix} \alpha_{0,0} & \alpha'_{0,1} & \cdots & \alpha'_{0,d-1} & 0 & \cdots & 0 \\ \alpha'_{0,1} & \alpha_{1,1} & \cdots & \alpha'_{1,d-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha'_{0,d-1} & \alpha'_{1,d-1} & \cdots & \alpha_{d-1,d-1} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} x \\ x^q \\ \vdots \\ x^{q^{n-1}} \end{bmatrix},$$

where $\alpha'_{i,j} = \frac{1}{2}\alpha_{i,j}$ and $d = \lceil \log_q(D) \rceil$. The KS-attack first interpolates an univariate representation of the public key over \mathbb{E} . This representation of the public key is isomorphic to the central map of Q-rank bounded by the ceiling of the logarithm of the degree bound. Thus, there is a linear map T^{-1} which when composed with the public key has Q-rank d , and so there is a low rank matrix that is an \mathbb{E} -linear combination of the Frobenius powers of G . This turns recovery of the transformation T into the solution of a MinRank problem over \mathbb{E} .

Another version of this attack, utilizing the same property, is the key recovery attack of [25]. The authors prove the existence of an \mathbb{E} -linear combination of the *public* key with low rank over \mathbb{E} . Setting the unknown coefficients of this linear combination as variables, they construct the ideal $I \subseteq R = \mathbb{F}[T]$ of minors of this sum of the appropriate dimension such that $V(I) \cap \mathbb{E}^{\dim(R)}$ consists of exactly such linear coefficients. Thus a Gröbner basis needs to be computed over \mathbb{F} and the variety computed over \mathbb{E} . This modeling of the KS-attack is called minors modeling and dramatically improves the efficiency of the KS-attack in many circumstances.

The KS-attack with either KS modeling or with minors modeling has also been used to break other HFE descendants. In [25], the minors modeling approach is used to break multi-HFE. In [15], the KS-attack is extended to provide key recovery for HFE-. In [14], both the KS modeling and minors modeling versions of the KS-attack are used to undermine the security of ZHFE.

The MinRank methodology is also employed in [16], where an effective key recovery attack on SRP is presented. It was shown that the low Q-rank of Square is exposed by the SRP construction. Specifically, the Q-rank of the square map $f(x) = x^2$ is one over an odd characteristic field. Since this low Q-rank map is in the span of the public polynomials, there is an \mathbb{E} -linear combination of the public polynomials of rank one! Thus the ideal generated by the two-by-two minors is resolved at degree two and the complexity of the attack is $\mathcal{O}\left(\binom{m+1}{2}^\omega\right)$, where $2 \leq \omega \leq 3$ is the linear algebra constant. The attack is applied practically, breaking the 80-bit parameters in about 8 min.

5 HFERP

In this section, we present a significant modification of SRP that we call HFERP. The key observation is that by replacing the Square map with a higher Q-rank instance of HFE, one can make the MinRank attack inefficient while maintaining

efficient inversion. For simplicity of the exposition, we present the scheme with a single layer UOV component, noting that it is trivial to replace UOV with a multi-layer Rainbow via the same construction.

Choose a finite field \mathbb{F}_q and let \mathbb{E} be a degree d extension field over \mathbb{F}_q . Let $\phi : \mathbb{F}_q^d \rightarrow \mathbb{E}$ be an \mathbb{F}_q -vector space isomorphism. Also, let o, r, s , and l be non-negative integers.

Key Generation. Let $n = d + o - l$, $n' = d + o$ and $m = d + o + r + s$. The central map of HFERP is the concatenation of an HFE core map, \mathcal{F}_{HFE} , an UOV (or alternatively, Rainbow) section, \mathcal{F}_R , and the plus modifier, \mathcal{F}_P . Formal definitions of the maps are provided below:

- The HFE component is defined as $\mathcal{F}_{HFE} : \mathbb{F}_q^{n'} \rightarrow \mathbb{F}_q^d$ and is the result of the following composition:

$$\mathbb{F}_q^{n'} \xrightarrow{\pi_d} \mathbb{F}_q^d \xrightarrow{\phi} \mathbb{E} \xrightarrow{f} \mathbb{E} \xrightarrow{\phi^{-1}} \mathbb{F}_q^d$$

where f is the HFE core map described in (1) and $\pi_d : \mathbb{F}_q^{d+o} \rightarrow \mathbb{F}_q^d$ is the projection onto the first d coordinates.

- The UOV (or alternatively, Rainbow) component is defined as

$$\mathcal{F}_R = (g^{(1)}, \dots, g^{(o+r)}) : \mathbb{F}_q^{n'} \rightarrow \mathbb{F}_q^{o+r}$$

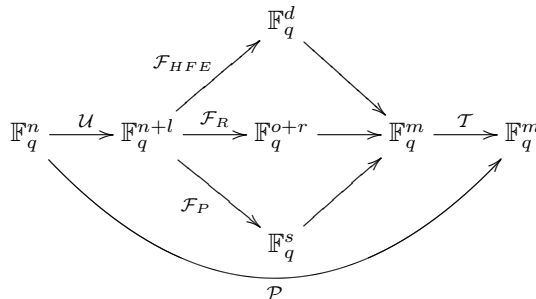
following the normal construction of the UOV signature scheme where $\mathcal{V} = \{1, \dots, d\}$ and $\mathcal{O} = \{d + 1, \dots, d + o\}$. For every $k \in \{1, \dots, o + r\}$, the quadratic polynomial $g^{(k)}$ is of the following form:

$$g^{(k)}(x_1, \dots, x_{n'}) = \sum_{i \in \mathcal{O}, j \in \mathcal{V}} \alpha^{(k)} x_i x_j + \sum_{i, j \in \mathcal{V}, i \leq j} \beta_{i,j}^{(k)} x_i x_j + \sum_{i \in \mathcal{V} \cup \mathcal{O}} \gamma_i^{(k)} x_i + \eta^{(k)}$$

where $\alpha^{(k)}$, $\beta_{i,j}^{(k)}$, $\gamma_i^{(k)}$, and $\eta^{(k)}$ are chosen at random from \mathbb{F}_q .

- The Plus modification is defined as $\mathcal{F}_P = (h^{(1)}, \dots, h^{(s)}) : \mathbb{F}_q^{n'} \rightarrow \mathbb{F}_q^s$ which consists of s randomly generated quadratic polynomials.

An affine embedding $\mathcal{U} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^{n'}$ of full rank and an affine isomorphism $\mathcal{T} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q^m$ are chosen for the butterfly construction as is common in big field schemes. The public key is given by $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{U} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$, where $\mathcal{F} = \mathcal{F}_{HFE} \parallel \mathcal{F}_R \parallel \mathcal{F}_P$ (\parallel being the concatenation function), and the private key is represented by the following figure:



Encryption. Given a message $M \in \mathbb{F}_q^n$, the ciphertext is computed as $C = \mathcal{P}(M) \in \mathbb{F}_q^m$.

Decryption. Given a ciphertext $C = (c_1, \dots, c_m) \in \mathbb{F}_q^m$, the decryption process is the following:

1. Compute $\mathbf{x} = (x_1, \dots, x_m) = \mathcal{T}^{-1}(C)$.
2. Compute $\mathbf{X} = \phi(x_1, \dots, x_d) \in \mathbb{E}$.
3. Use the Berlekamp algorithm to compute the inverse of the HFE polynomials to recover $\mathbf{y} = (y_1, \dots, y_d)$.
4. Given the vinegar values y_1, \dots, y_d , solve the system of $o + r$ linear equations in the $n' - d = o$ variables $u_{d+1}, \dots, u_{n'}$ given by

$$g^{(k)}(y_1, \dots, y_d, u_{d+1}, \dots, u_{n'}) = x_{d+k}$$

for $k = 1, \dots, o + r$. The solution is denoted $(y_{d+1}, \dots, y_{n'})$.

5. Compute the plaintext $M \in \mathbb{F}_q^n$ by finding the preimage of $(y_1, \dots, y_{n'})$ under the affine embedding \mathcal{U} .

6 Complexity of Attack

In this section we derive tight complexity estimates or proofs of resistance for the principal relevant attacks on HFERP. These attacks include the direct algebraic attack, the MinRank attack, the small field MinRank and dual rank attacks, and the invariant attack.

6.1 Algebraic Attack

The algebraic attack attempts to invert the public key at a ciphertext directly via the calculation of a Gröbner basis. It is commonly believed that the closeness of the solving degree of a polynomial system, the degree at which the Gröbner basis is resolved, and the degree of regularity, the degree at which a non-trivial syzygy producing a degree fall first occurs, is a generic property. Thus the lower bound on the complexity of the algebraic attack that the degree of regularity provides is likely a tight bound, and is consequently a critical quantity for analyzing the security of the scheme.

Theorem 1. *The degree of regularity of the public key of HFERP is bounded by*

$$d_{reg} \leq \begin{cases} \frac{(q-1)\lceil \log_q(D) \rceil}{2} + 2 & \text{if } q \text{ is odd or } \lceil \log_q(D) \rceil \text{ is even,} \\ \frac{(q-1)(\lceil \log_q(D) \rceil + 1)}{2} + 1 & \text{otherwise.} \end{cases}$$

Proof. There is a linear function of the public key separating the HFE polynomials \mathcal{H} from the non-HFE polynomials \mathcal{N} . Trivially, the d_{reg} is bounded by the degree of regularity of the system \mathcal{H} , which, via [26, Theorem 4.2], produces the above bound.

One must note that the above bound is not what is needed to ensure security. Instead we require a lower bound. Extensive experimentation shows that for very small q , the above estimate is tight. We have, however, a further complication. In general, adding more polynomials to an ideal may decrease its degree of regularity. To address this issue we have conducted small scale experiments showing that the degree of regularity and solving degree behave similarly to those of random systems, see Sect. 7.

Conjecture 1. *Under the assumption that the degree of regularity is at least $\lceil \log_q(D) \rceil + 2$ for small odd q and sufficiently large n , the complexity of the algebraic attack is given by*

$$Comp.alg = \mathcal{O} \left(\binom{n + d_{reg}}{d_{reg}}^2 \binom{n}{2} \right) = \mathcal{O} \left(n^{2\lceil \log_q(D) \rceil + 6} \right).$$

6.2 MinRank Attack

The min-rank attack proposed in [16] is so successful due to the Q-rank of the squaring map within SRP being equal to one. By changing the square map component to an HFE core map, we are able to thwart such an attack on HFERP. This subsection walks through the attack proposed in [16], with HFERP in mind, and proves that the min-Q-rank of HFERP differs from SRP.

Note that, similar to SRP, the public key of HFERP has an analogous scheme without embedding as long as $\pi_d \circ \mathcal{U}$ is of full rank, which it is defined to be in this scheme. Let $\pi'_d : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^d$ be the projection onto the first d coordinates and find a projection $\rho : \mathbb{F}_q^{n+l} \rightarrow \mathbb{F}_q^n$ such that $\mathcal{U}' = \rho \circ \mathcal{U}$ has full rank and $\pi'_d \circ \mathcal{U}' = \pi_d \circ \mathcal{U}$. Let $\mathcal{F}^* : \mathbb{E} \rightarrow \mathbb{E}$ represent the chosen high Q-rank HFE core map so that $\mathcal{F}_{HFE} = \phi^{-1} \circ \mathcal{F}^* \circ \phi \circ \pi_d$. Then identify the Rainbow and random components as $\mathcal{F}'_R : \mathcal{F}_R \circ \mathcal{U} \circ \mathcal{U}'^{-1}$ and $\mathcal{F}'_P : \mathcal{F}_P \circ \mathcal{U} \circ \mathcal{U}'^{-1}$ respectively. Thus, one can see that

$$\mathcal{T} \circ \begin{bmatrix} \phi \circ \mathcal{F}^* \circ \phi^{-1} \circ \pi_d \\ \mathcal{F}_R \\ \mathcal{F}_P \end{bmatrix} \circ \mathcal{U} = \mathcal{T} \circ \begin{bmatrix} \phi \circ \mathcal{F}^* \circ \phi^{-1} \circ \pi'_d \\ \mathcal{F}'_R \\ \mathcal{F}'_P \end{bmatrix} \circ \mathcal{U}'.$$

Notice that the attack on SRP was not just a min-rank attack on the public key of SRP, but on a linear combination of public forms of SRP that had low Q-rank over the degree d extension used by the squaring component. This method allowed the attack to ignore the fact that the public key of an instance of SRP was expected to be of high rank. Thus, to demonstrate that HFERP resists such an attack, we briefly outline the method of deriving the linear combination of public forms from [16] for HFERP and prove that the min-Q-Rank of the result is sufficiently high to resist such an attack.

Let α be a primitive element of the degree d extension \mathbb{E} of \mathbb{F}_q . Fix a vector space isomorphism $\phi : \mathbb{F}_q^d \rightarrow \mathbb{E}$ defined by $\phi(\bar{x}) = \sum_{i=0}^{d-1} x_i \alpha^i$. Then, fix a one dimensional representation $\Phi : \mathbb{E} \rightarrow \mathbb{A}$ defined by $a \mapsto (a, a^q, \dots, a^{q^{d-1}})$. Next,

define $\mathcal{M}_d : \mathbb{F}_q^d \rightarrow \mathbb{A}$ by $\mathcal{M}_d = \Phi \circ \phi$. It was demonstrated you can look at this map through the following matrix representation

$$\mathbf{M}_d = \begin{bmatrix} 1 & 1 & \dots & 1 \\ \alpha & \alpha^q & \dots & \alpha^{q^{d-1}} \\ \alpha^2 & \alpha^{2q} & \dots & \alpha^{2q^{d-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{d-1} & \alpha^{(d-1)q} & \dots & \alpha^{(d-1)q^{d-1}} \end{bmatrix} \in \mathcal{M}_{d \times d}(\mathbb{E})$$

This matrix allows the passage from \mathbb{F}_q^d and \mathbb{A} easily by right multiplication with \mathbf{M}_d or \mathbf{M}_d^{-1} . Next are a few more definitions necessary to be able to look at a matrix representation of the public key:

$$\widetilde{\mathbf{M}}_d = \begin{bmatrix} \mathbf{M}_d & 0 \\ 0 & \mathbf{I}_{o+r+s} \end{bmatrix} \in \mathcal{M}_{m \times m}(\mathbb{E})$$

$$\widehat{\mathbf{M}}_d = \begin{bmatrix} \mathbf{M}_d \\ \mathbf{0}_{o \times d} \end{bmatrix} \in \mathcal{M}_{(d+o) \times d}(\mathbb{E})$$

Finally, define \mathbf{F}^{*i} be the matrix representation of the quadratic form over \mathbb{A} of the i^{th} Frobenius power of the chosen HFE core map. Now we have all the necessary notation to view the public key as a matrix equation.

Denote the m -dimensional vector of $(d + o) \times (d + o)$ symmetric matrices associated by the private key as follows:

$$(\mathbf{F}_{(HFE,0)}, \dots, \mathbf{F}_{(HFE,d-1)}, \mathbf{F}_{(R,0)}, \dots, \mathbf{F}_{(R,o+r-1)}, \mathbf{F}_{(P,0)}, \dots, \mathbf{F}_{(P,s-1)}). \tag{2}$$

Note that the function corresponding to the application of each coordinate of a vector of the quadratic forms followed by the application of a linear map represented by a matrix is denoted as a right product of the vector and a matrix representation of the linear map.

Next, observe

$$(\mathbf{F}_{(HFE,0)}, \dots, \mathbf{F}_{(HFE,d-1)})\mathbf{M}_d = (\widehat{\mathbf{M}}_d \mathbf{F}^{*0} \widehat{\mathbf{M}}_d^\top, \dots, \widehat{\mathbf{M}}_d \mathbf{F}^{*(d-1)} \widehat{\mathbf{M}}_d^\top),$$

which yields

$$(\bar{x} \mathbf{F}_{(HFE,0)} \bar{x}^\top, \dots, \bar{x} \mathbf{F}_{(HFE,d-1)} \bar{x}^\top) \mathbf{M}_d = (\bar{x} \widehat{\mathbf{M}}_d \mathbf{F}^{*0} \widehat{\mathbf{M}}_d^\top \bar{x}^\top, \dots, \bar{x} \widehat{\mathbf{M}}_d \mathbf{F}^{*(d-1)} \widehat{\mathbf{M}}_d^\top \bar{x}^\top),$$

as a function of \bar{x} . This gives the following equation:

$$(\mathbf{F}_{(HFE,0)}, \dots, \mathbf{F}_{(HFE,d-1)}, \mathbf{F}_{(R,0)}, \dots, \mathbf{F}_{(P,s-1)}) \widetilde{\mathbf{M}}_d = (\widehat{\mathbf{M}}_d \mathbf{F}^{*0} \widehat{\mathbf{M}}_d^\top, \dots, \widehat{\mathbf{M}}_d \mathbf{F}^{*(d-1)} \widehat{\mathbf{M}}_d^\top, \mathbf{F}_{(R,0)}, \dots, \mathbf{F}_{(P,s-1)}) \tag{3}$$

Now, look to the relation between the public key and its corresponding private key central maps:

$$(\mathbf{P}_0, \dots, \mathbf{P}_{m-1}) \mathbf{T}^{-1} = (\mathbf{U} \mathbf{F}_{(HFE,0)} \mathbf{U}^\top, \dots, \mathbf{U} \mathbf{F}_{(P,s-1)} \mathbf{U}^\top). \tag{4}$$

By combining Eqs. 3 and 4, we have the following:

$$\begin{aligned}
 & (\mathbf{P}_0, \dots, \mathbf{P}_{m-1}) \mathbf{T}^{-1} \widetilde{\mathbf{M}}_d = \\
 & (\mathbf{U} \widehat{\mathbf{M}}_d \mathbf{F}^{*0} \widehat{\mathbf{M}}_d^\top \mathbf{U}^\top, \dots, \mathbf{U} \widehat{\mathbf{M}}_d \mathbf{F}^{*(d-1)} \widehat{\mathbf{M}}_d^\top \mathbf{U}^\top, \mathbf{U} \mathbf{F}_{(R,0)} \mathbf{U}^\top, \dots, \mathbf{U} \mathbf{F}_{(P,s-1)} \mathbf{U}^\top)
 \end{aligned}$$

As in [16], let $\widehat{\mathbf{T}} = \mathbf{T}^{-1} \widetilde{\mathbf{M}}_d = [t_{i,j}] \in \mathcal{M}_{m \times m}(\mathbb{E})$ and $\mathbf{W} = \mathbf{U} \widehat{\mathbf{M}}_d$. This identification produces

$$\sum_{i=0}^{m-1} t_{i,0} \mathbf{P}_i = \mathbf{W} \mathbf{F}^{*0} \mathbf{W}^\top. \tag{5}$$

Since the rank of \mathbf{F}^{*i} is equal to the Q-rank of the quadratic form of the HFE core map for all i , the rank of this \mathbb{E} -linear combination of the public matrices is bounded by the minimum of the rank of $\mathbf{U} \widehat{\mathbf{M}}_d$ and the rank of \mathbf{F}^{*0} , *id est* the Q-rank of our HFE core map. This statement forms the following theorem:

Theorem 2. *The min-Q-rank of the public key P of HFERP(q, d, o, r, s, l) is given by:*

$$\text{min-Q-rank}(P) \leq \min\{\text{Rank}(\mathbf{U} \widehat{\mathbf{M}}_d), \text{Rank}(\mathbf{F}^{*0})\}$$

Proof. The proof in [16] describes the parameters in which the min-Q-rank(P) can be equal to zero. So, we move forward with the assumption that $\mathbf{U} \widehat{\mathbf{M}}_d \neq 0$, which occurs with high probability when $d > l$. In (5) we have a linear combination of the public key equations equal to the following:

$$\mathbf{W} \mathbf{F}^{*0} \mathbf{W}^\top = \mathbf{U} \widehat{\mathbf{M}}_d \mathbf{F}^{*0} \widehat{\mathbf{M}}_d^\top \mathbf{U}^\top. \tag{6}$$

This proves our result.

It should be noted that \mathbf{U} , $\widehat{\mathbf{M}}_d$, and \mathbf{F}^{*0} are chosen by the user. They can easily be chosen in such a way such that

$$\text{min-Q-rank}(P) = \min\{\text{Rank}(\mathbf{U} \widehat{\mathbf{M}}_d), \text{Rank}(\mathbf{F}^{*0})\}.$$

This would also occur with high probability if \mathbf{U} , $\widehat{\mathbf{M}}_d$, and \mathbf{F}^{*0} were randomly generated. Directly from [15], we also have the following complexity for the MinRank attack on HFERP:

Corollary 1. *The complexity of the MinRank attack with minors modeling on HFERP is given by*

$$\text{Comp.Minors} = \mathcal{O} \left(\binom{m + \lfloor \log_q(D) \rfloor}{\lfloor \log_q(D) \rfloor}^2 \binom{m}{2} \right) = \mathcal{O} \left(m^{2 \lfloor \log_q(D) \rfloor + 2} \right).$$

6.3 Base-Field Rank and Invariant Attacks

Variants of several attacks applicable to other versions of the Rainbow cryptosystem are applicable to HFERP. These include the linear-algebra-search version of MinRank [27], the HighRank attack [27] and the UOV invariant attack [4].

The MinRank attack works by randomly choosing one or more vectors \mathbf{w}_j in the plaintext space and solving for a linear combination $t_i \in \mathbb{F}$ of the plaintext equations satisfying:

$$\sum_{i=1}^m t_i Df_i(\mathbf{w}_j) = 0$$

The attack succeeds when \mathbf{w}_j is in the kernel of a low rank linear combination of differentials of the public polynomials. In the case of HFERP, the HFE component equations form a d -dimensional subspace of the public equations having rank d over \mathbb{F} . Note that the attacker can remove up to $d-1$ equations while preserving at least a one dimensional subspace of low rank maps. Thus, the attack can succeed with a one dimensional solution space for t_i and only a single \mathbf{w}_j as long as $m \leq n + d$.

If $m > n + d$, the adversary may still use a single vector \mathbf{w}_j to constrain the t_i 's rather than attempting to find two vectors in the kernel of the HFE equations. In this case, the attacker must search through an $m - n - d + 1$ dimensional space of spurious solutions to find the useful 1 dimensional space of t_i s. This method is still less expensive than searching for two vectors in the kernel of the HFE equations when $m < n + 2d$.

It should be further noted that, since the differentials of the oil maps will map any vector in the kernel of the HFE equations to the d -dimensional HFE input space, we expect an $o_1 + r_1 - d$ dimensional subspace of the oil equations to also have such a vector in the kernel of their differentials, see Fig. 1. Thus, when $m < n + \max(d, o_1 + r_1)$, vectors in the HFE kernel can be recognized, because they are in the kernel of an unusually large subspace of the public equations, and when $2d < n$ the linear combinations of the public equations from the HFE and oil spaces can be recognized due to their low rank.

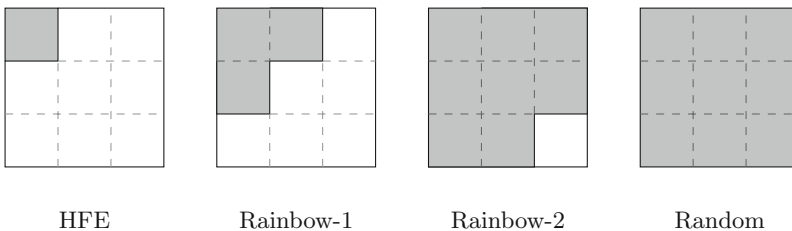


Fig. 1. The shape of the matrix representations of the central maps of HFERP. The shaded regions represent possibly nonzero values while unshaded areas have coefficients of zero.

Thus the complexity of MinRank (for plausible choices of m) is

$$\text{Comp. MinRank} = \begin{cases} \mathcal{O}(q^d m^\omega) & m < n + \max(d, o_1 + r_1) \\ \mathcal{O}(q^{d+m-n-\max(d, o_1+r_1)} n^\omega) & \begin{array}{l} m \geq n + \max(d, o_1 + r_1) \\ m < n + d + \max(d, o_1 + r_1) \\ n > 2d \end{array} \\ \mathcal{O}(q^{m-n} n^\omega) & \begin{array}{l} m \geq n + \max(d, o_1 + r_1) \\ m < n + 2d \\ n \leq 2d \end{array} \\ \mathcal{O}(q^{2d} m^\omega) & \begin{array}{l} m < 2n + \max(d, o_1 + r_1 - d) \\ \text{No better attack.} \end{array} \end{cases}$$

In the HighRank attack, the attacker randomly selects linear combinations of the public polynomials with the hope of selecting a polynomial with significantly less than full rank. This attack takes advantage of the $d + o_1 + r_1$ -dimensional subspace of the public polynomials generated by the HFE maps and either the Rainbow-1 maps of Fig. 1 or for UOV of the d -dimensional HFE subspace. The complexity of the attack is then:

$$\text{Comp. HighRank} = \mathcal{O}(q^{m-d-o_1-r_1} n^\omega).$$

It should also be noted that linear combinations of HFE and Rainbow-1 polynomials form an $m - s$ dimensional subspace of the public polynomials, that act linearly on the $o_2 - l$ dimensional preimage under \mathcal{U} of the oil subspace. This bounds their rank to be at most $2d$. Noting that the probability that a random square matrix has corank a is approximately q^{-a^2} , we see that, the high rank attack can be straightforwardly applied if $2d < n - \sqrt{m - d - o_1 - r_1}$.

Additionally, the HighRank attack can be combined with the oil and vinegar invariant attack to distinguish linear combinations of the HFE and Rainbow maps from other linear combinations of the public maps. Here, a pair of maps from the HFE and Rainbow subspace can be identified by restricting their differentials to a subspace of the plaintext space in which both maps are full rank, and checking to see if $(Dp_1)^{-1} Dp_2$ has a large invariant subspace (which will be the intersection of the preimage of the oil subspace under \mathcal{U} and the subspace used to restrict the differentials). This allows the high rank attack to be applied with similar complexity as long as $2d < n - \sqrt{\frac{m-d-o_1-r_1}{2}}$: Applying the attack will involve testing no more than $\left(q^{\frac{m-d-o_1-r_1}{2}}\right)^2 = q^{m-d-o_1-r_1}$ pairs of rank $n - 2d$ maps, and therefore this step will not dominate the complexity of the approximately $q^{m-d-o_1-r_1}$ rank computations involved in the HighRank step.

If $2d \geq \zeta$, where $\zeta_1 = n - \sqrt{\frac{m-d-o_1-r_1}{2}}$, the complexity of HighRank is given by:

$$\text{Comp. HighRank} = \begin{cases} \text{Comp. HighRank} = \mathcal{O}(q^{m-d}n^\omega) & 2d \geq \zeta_1 \\ \text{Comp. HighRank} = \mathcal{O}(q^{m-d-o_1-r_1}n^\omega) & 2d < \zeta_1. \end{cases}$$

Finally, when $2d \geq n - \sqrt{\frac{m-d-o_1-r_1}{2}}$, as in the UOV attack, the previous steps must be combined with a projection, aimed at removing enough vinegar variables that the restriction of the differentials of linear combinations of HFE and Rainbow maps to the projected plaintext space is less than full rank. This yields a complexity for hybrid HighRank/UOV invariant type attacks of:

$$\text{Comp. UOV} = \begin{cases} \mathcal{O}(q^{m-d-o_1-r_1}n^\omega) & n > \zeta_2 \\ \mathcal{O}\left(q^{m-d-o_1-r_1+\sqrt{\frac{m-d-o_1-r_1}{2}+2d-n}(o_1+o_2-l)^4}\right) & n \leq \zeta_2. \end{cases}$$

where $\zeta_2 = 2d + \sqrt{\frac{m-d-o_1-r_1}{2}}$. This attack may also be applied to the Rainbow-2 maps of Fig. 1 in which case the complexity is:

$$\text{Comp. UOV2} = \begin{cases} \mathcal{O}(q^s n^\omega) & n > 2d + 2o_1 + \sqrt{\frac{s}{2}} \\ \mathcal{O}\left(q^{s+\sqrt{\frac{s}{2}+2d+2o_1-n}(o_2-l)^4}\right) & n \leq 2d + 2o_1 + \sqrt{\frac{s}{2}}. \end{cases}$$

7 Parameter Selection and Experimental Results

We propose single-layer parameters (A) and (B) for 80-bit security and multi-layer parameters (C) and (D) for 128-bit security:

- (A) $(q = 3, d = 42, o = 21, r = 15, s = 17, l = 0, D = 3^7 + 1)$
- (B) $(q = 3, d = 63, o = 21, r = 11, s = 10, l = 0, D = 3^7 + 1)$
- (C) $(q = 3, d = 85, o_1 = o_2 = 70, r_1 = r_2 = 89, s = 61, l = 0, D = 3^7 + 1)$
- (D) $(q = 3, d = 60, o_1 = o_2 = 40, r_1 = r_2 = 23, s = 40, l = 0, D = 3^9 + 1)$

Then we have the following values for (n, m) : (63, 95) for (A), (84, 105) for (B), (225, 464) for (C), and (140, 226) for (D). The security level for suggested parameters is estimated by all the attack in Sect. 6. Here, we assume that the degree of regularity for direct attack is 10 by eecture 1 for (A), (B), and (C) while it is 12 for (D).

To draw a direct comparison with HFE, note that to achieve the same security level as HFERP, an HFE scheme requires m equations, and hence $n = m$ variables. Therefore secure HFE public keys are far larger while offering slower decryption due to the use of the Berlekamp algorithm in a far larger field.

We ran a series of experiments with Magma, see [28], on a 2.6 GHz Intel[®] Xeon^R CPU¹. These are not optimized implementations (Table 1).

Table 1. Experimental results for HFERP.

	(A)	(B)	(C)	(D)
Key Generation	0.299 s	0.572 s	20.498 s	3.43 s
Encryption	0.001 s	0.001 s	0.006 s	0.001 s
Decryption	3.977 s	8.671 s	49.182 s	124.27 s
Secret Key Size	19.8 KB	31.7 KB	1344.0 KB	226.0 KB
Public Key Size	48.2 KB	93.6 KB	2905.7 KB	552.3 KB

We also investigated the growth of the first fall degree (d_{reg}) as well as the solving degree with five experiments performed at each of eight different parameters sets. We directly compared these data with randomly generated systems, see Table 2.

Table 2. Direct attack experiment data for various values of d, o, r, s . (s.r.d. stands for semi- regular degree)

(q, d, o, r, s, l, D)	n	m	HFERP		Random		
			d_{reg}	sol. deg	d_{reg}	sol. deg	s.r.d.
A. Direct Attack, $d = 2o, d + o \equiv 2(r + s), o = 4, 5, 6, 7$							
$(3, 8, 4, 3, 3, 0, 2188)$	12	18	4, 4, 4, 4, 4	4, 4, 4, 4, 4	4, 4, 4, 4, 4	4, 4, 4, 4, 4	4
$(3, 10, 5, 4, 3, 0, 2188)$	15	22	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5
$(3, 12, 6, 5, 4, 0, 2188)$	18	27	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5
$(3, 14, 7, 5, 5, 0, 2188)$	21	31	6, 5, 5, 5, 5	6, 6, 6, 6, 6	5, 5, 5, 5, 5	6, 6, 6, 6, 6	6
B. Direct Attack, $d = 3o, r + s \equiv o, o = 3, 4, 5, 6$							
$(3, 9, 3, 2, 2, 0, 2188)$	12	16	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5
$(3, 12, 4, 2, 2, 0, 2188)$	16	20	5, 6, 6, 5, 5	5, 6, 6, 6, 5	6, 5, 6, 6, 5	6, 6, 6, 6, 6	6
$(3, 15, 5, 3, 3, 0, 2188)$	20	26	6, 5, 5, 5, 5	6, 6, 6, 6, 6	5, 5, 5, 6, 5	6, 6, 6, 6, 6	6
$(3, 18, 6, 3, 3, 0, 2188)$	24	30	5, 5, 5, 5, 5	7, 7, 7, 7, 7	5, 5, 5, 5, 7	7, 7, 7, 7, 7	7
(d, o, r, s, l, D)	n	m	HFERP		Random		
			d_{reg}	sol. deg	d_{reg}	sol. deg	s.r.d.
C. Direct Attack, $d \equiv 3.4a, o \equiv (2.8a, 2.8a), r \equiv (3.56a, 3.56a), s \equiv 2.44a, a = 1, 2, 3, 4$							
$(3, (3, 3), (4, 4), 2, 0, 2188)$	9	19	3, 3, 3, 3, 3	3, 3, 2, 3, 2	3, 3, 3, 3, 3	2, 3, 3, 2, 2	3
$(7, (6, 6), (7, 7), 5, 0, 2188)$	19	38	4, 4, 4, 4, 4	4, 4, 4, 4, 4	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5
$(10, (8, 8), (11, 11), 7, 0, 2188)$	26	55	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5
$(14, (11, 11), (14, 14), 10, 0, 2188)$	36	74	5		5		6
D. Direct Attack, $d \equiv 2.4a, o \equiv (1.6a, 1.6a), r \equiv (0.92a, 0.92a), s \equiv 1.6a, a = 2, 3, 4, 5$							
$(5, (3, 3), (2, 2), 3, 0, 3^9 + 1)$	11	18	4, 4, 4, 4, 4	4, 4, 4, 4, 4	4, 4, 4, 4, 4	4, 4, 4, 3, 4	4
$(7, (5, 5), (3, 3), 5, 0, 3^9 + 1)$	17	28	4, 4, 4, 4, 4	4, 4, 4, 4, 4	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5
$(10, (6, 6), (4, 4), 6, 0, 3^9 + 1)$	22	36	5, 5, 5, 5, 5	5, 5, 5, 5, 5	5, 5, 5, 5, 5	6, 6, 6, 6, 6	6
$(12, (8, 8), (5, 5), 8, 0, 3^9 + 1)$	28	46	5, 5	6, 6	5, 5	6	6

¹ Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

For comparison, we include the semi-regular degree for systems of m equations in n variables. This quantity was calculated by computing the first non-positive coefficient in the series

$$S_{n,m}(t) = \frac{(1-t^q)^n(1-t^2)^m}{(1-t)^n(1-t^{2q})^m}.$$

Noting that the degree of regularity of the zero-dimensional ideal is the same as the first fall degree of the ideal generated by the homogeneous components of the generators of highest degree. We derive the above formula as the fusion of the techniques in [29, 30].

It is clear that the degree of regularity of the small scale instances of HFERP grows in relation to that of random schemes. By the data in the tables, we can estimate that the degree of regularity for direct attack on (A) and (B) is greater than 9 at least.

8 Conclusion

SRP was an ambitious encryption scheme attempting to combine the efficiency of the inversion of Square with the security of Rainbow to achieve security with a small blow-up factor between the plaintext and ciphertext. Unfortunately, this technique was a bit too ambitious.

Interestingly, the idea of replacing Square with a more general and higher Q-rank HFE primitive seems to solve this problem. Even more interestingly, the resulting scheme, HFERP, though in principle assailable via essentially every major cryptanalytic technique available in multivariate cryptography, appears to be out of range of these myriad attacks.

The parameter ℓ in SRP was introduced for efficiency, attempting to reduce the public key size while maintaining the algebraic structure of the scheme. We have found that this quantity adds nothing to security and have set it equal to zero for our suggested parameters. An interesting possible future problem is to determine whether ℓ can be securely set to a value larger than zero and thereby reduce public key size. For now, we err on the side of caution, and conservatively use all of the entropy we can get.

Acknowledgments. The first and fourth authors were supported by JST CREST (Grant Number JPMJCR14D6).

A Toy Example

The purpose of the following toy example is to help the reader understand the process of generating a public key for an instance of HFERP as well as an example of encryption and decryption. The parameters used are by no means secure and are solely for instructional purposes.

Parameters of this toy example are as follows: $q = 7$, $d = o = r = 2$, $s = 1$, and $l = 0$. Then, construct \mathbb{E} a degree 2 extension field over \mathbb{F}_7 . The chosen

HFE core map is $f = \xi^{12}X^{14} + \xi^6X^8 + \xi^{29}X^2$ where $\xi \in \mathbb{E}$. Let \mathcal{T} and \mathcal{U} be the following affine maps:

$$\mathcal{T} = \begin{bmatrix} 2 & 1 & 2 & 4 & 5 & 0 & 3 \\ 1 & 1 & 3 & 3 & 4 & 4 & 4 \\ 4 & 2 & 1 & 3 & 1 & 0 & 6 \\ 0 & 1 & 0 & 1 & 5 & 5 & 5 \\ 5 & 5 & 3 & 6 & 4 & 2 & 4 \\ 2 & 5 & 1 & 6 & 5 & 6 & 0 \\ 1 & 1 & 2 & 2 & 6 & 4 & 3 \end{bmatrix}, \mathcal{U} = \begin{bmatrix} 4 & 6 & 6 & 4 \\ 3 & 2 & 0 & 2 \\ 1 & 1 & 6 & 5 \\ 3 & 6 & 6 & 6 \end{bmatrix}$$

With the parameters described above, \mathcal{F} can be represented as the following matrices over \mathbb{F}_7

$$F_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, F_2 = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 1 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, F_3 = \begin{bmatrix} 3 & 1 & 6 & 1 \\ 3 & 1 & 4 & 5 \\ 3 & 4 & 0 & 0 \\ 3 & 2 & 0 & 0 \end{bmatrix},$$

$$F_4 = \begin{bmatrix} 5 & 1 & 0 & 3 \\ 0 & 5 & 0 & 3 \\ 0 & 4 & 0 & 0 \\ 6 & 1 & 0 & 0 \end{bmatrix}, F_5 = \begin{bmatrix} 6 & 0 & 3 & 4 \\ 6 & 2 & 4 & 2 \\ 6 & 3 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}, F_6 = \begin{bmatrix} 4 & 4 & 1 & 1 \\ 3 & 0 & 0 & 3 \\ 3 & 6 & 0 & 0 \\ 1 & 2 & 0 & 0 \end{bmatrix}, F_7 = \begin{bmatrix} 6 & 3 & 2 & 3 \\ 4 & 4 & 0 & 6 \\ 2 & 3 & 1 & 3 \\ 6 & 4 & 0 & 6 \end{bmatrix}$$

P_1 and P_2 represent the HFE component, $P_3 \rightarrow P_6$ represent the rainbow component, and P_7 represents the plus component. With the public key generated by $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{U}$, its matrix form over \mathbb{F}_7 is:

$$P_1 = \begin{bmatrix} 1 & 1 & 2 & 5 \\ 1 & 2 & 3 & 2 \\ 3 & 2 & 4 & 4 \\ 3 & 3 & 0 & 3 \end{bmatrix}, P_2 = \begin{bmatrix} 0 & 2 & 0 & 6 \\ 4 & 5 & 2 & 0 \\ 6 & 3 & 3 & 4 \\ 3 & 1 & 2 & 2 \end{bmatrix}, P_3 = \begin{bmatrix} 2 & 3 & 1 & 4 \\ 4 & 5 & 4 & 5 \\ 3 & 5 & 5 & 1 \\ 5 & 1 & 0 & 6 \end{bmatrix},$$

$$P_4 = \begin{bmatrix} 0 & 6 & 0 & 2 \\ 1 & 3 & 0 & 2 \\ 5 & 1 & 5 & 1 \\ 5 & 3 & 0 & 5 \end{bmatrix}, P_5 = \begin{bmatrix} 4 & 3 & 2 & 3 \\ 6 & 5 & 2 & 4 \\ 4 & 3 & 1 & 5 \\ 5 & 2 & 4 & 5 \end{bmatrix}, P_6 = \begin{bmatrix} 1 & 4 & 2 & 2 \\ 3 & 3 & 6 & 2 \\ 5 & 4 & 0 & 0 \\ 3 & 5 & 5 & 4 \end{bmatrix}, P_7 = \begin{bmatrix} 1 & 3 & 6 & 0 \\ 0 & 3 & 4 & 0 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 6 & 4 \end{bmatrix}$$

Given the following plaintext, $(2, 6, 1, 5)$, the resulting ciphertext is $(0, 0, 1, 3, 0, 4, 0)$.

Decryption: Given a ciphertext $(0, 0, 1, 3, 0, 4, 0)$, the following process is how you would obtain its corresponding plaintext. Part of the secret key:

$$\mathcal{T}^{-1} = \begin{bmatrix} 1 & 6 & 4 & 2 & 2 & 2 & 5 \\ 5 & 4 & 4 & 6 & 0 & 5 & 2 \\ 5 & 3 & 5 & 2 & 3 & 2 & 4 \\ 5 & 6 & 5 & 5 & 2 & 1 & 1 \\ 2 & 5 & 4 & 2 & 1 & 5 & 2 \\ 2 & 5 & 6 & 6 & 3 & 5 & 5 \\ 1 & 2 & 5 & 4 & 4 & 0 & 5 \end{bmatrix}, \mathcal{U}^{-1} = \begin{bmatrix} 4 & 5 & 2 & 1 \\ 3 & 1 & 3 & 1 \\ 4 & 1 & 2 & 0 \\ 5 & 6 & 1 & 1 \end{bmatrix}$$

Feed the ciphertext through \mathcal{T}^{-1} to get

$$(0, 6, 2, 6, 0, 4, 6) \tag{7}$$

The first $d = 2$ elements are the corresponding HFE outputs. Take these elements and adjust the HFE core map as follows:

$$f := f - 0\xi^{1-1} - 6\xi^{2-1} = \xi^{12}X^{14} + \xi^6X^8 + \xi^{29}X^2 + \xi$$

Perform the Berlekamp algorithm to find the preimage of f . In doing so in this toy example, you get $(0, 6)$. Next, construct the vector:

$$\bar{u} = [0, 6, u_1, u_2].$$

Construct equations of the form $\bar{u}F_1\bar{u}^\top = x_i$ where x_i refers to the i^{th} element of (7), for $i \in \{3, 4, 5, 6\}$. This will result with the following equations:

$$\begin{bmatrix} 6u_1 + 1 \\ 3u_1 + 3u_2 + 5 \\ 2u_2 + 2 \\ u_1 + 2u_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 0 \\ 4 \end{bmatrix}$$

Solving this system of equations gives us $u_1 = 6$ and $u_2 = 6$. Thus,

$$\bar{u} = [0, 6, 6, 6].$$

Finally, feed this through \mathcal{U}^{-1} to get the plaintext, $[2, 6, 1, 5]$.

References

1. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Stat. Comput.* **26**, 1484 (1997)
2. Mosca, M.: Cybersecurity in a quantum world: will we be ready? In: Workshop on Cybersecurity in a Post-Quantum World, Invited Presentation (2015). <https://csrc.nist.gov/csrc/media/events/workshop-on-cybersecurity-in-a-post-quantum-world/documents/presentations/session8-mosca-michele.pdf>
3. Yasuda, T., Sakurai, K.: A multivariate encryption scheme with rainbow. In: Qing, S., Okamoto, E., Kim, K., Liu, D. (eds.) *ICICS 2015*. LNCS, vol. 9543, pp. 236–251. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29814-6_19
4. 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
5. Ding, J., Schmidt, D.: Rainbow, a new multivariable polynomial signature scheme. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) *ACNS 2005*. LNCS, vol. 3531, pp. 164–175. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_12
6. 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

7. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_4
8. Tao, C., Diene, A., Tang, S., Ding, J.: Simple matrix scheme for encryption. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 231–242. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_16
9. Ding, J., Petzoldt, A., Wang, L.: The cubic simple matrix encryption scheme. [32], pp. 76–87 (2014)
10. Porras, J., Baena, J., Ding, J.: ZHFE, A new multivariate public key encryption scheme. [32], pp. 229–245 (2014)
11. Moody, D., Perlner, R.A., Smith-Tone, D.: An asymptotically optimal structural attack on the ABC multivariate encryption scheme. [32], pp. 180–196 (2014)
12. Moody, D., Perlner, R., Smith-Tone, D.: Key recovery attack on the cubic ABC simple matrix multivariate encryption scheme. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 543–558. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_29
13. Moody, D., Perlner, R.A., Smith-Tone, D.: Improved attacks for characteristic-2 parameters of the cubic ABC simple matrix encryption scheme. [31], pp. 255–271 (2017)
14. Cabarcas, D., Smith-Tone, D., Verbel, J.A.: Key recovery attack for ZHFE. [31], pp. 289–308 (2017)
15. Vates, J., Smith-Tone, D.: Key recovery attack for all parameters of HFE-. [31], pp. 272–288 (2017)
16. Perlner, R., Petzoldt, A., Smith-Tone, D.: Total break of the SRP encryption scheme. In: Adams, C., Camenisch, J. (eds.) SAC 2017. LNCS, vol. 10719, pp. 355–373. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72565-9_18
17. Matsumoto, T., Imai, H.: Public quadratic polynomial-tuples for efficient signature-verification and message-encryption. In: Barstow, D., Brauer, W., Brinch Hansen, P., Gries, D., Luckham, D., Moler, C., Pnueli, A., Seegmüller, G., Stoer, J., Wirth, N., Günther, C.G. (eds.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-45961-8_39
18. Berlekamp, E.R.: Factoring polynomials over large finite fields. *Math. Comput.* **24**, 713–735 (1970)
19. Clough, C., Baena, J., Ding, J., Yang, B.-Y., Chen, M.: Square, a new multivariate encryption scheme. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 252–264. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00862-7_17
20. Patarin, J.: The oil and vinegar algorithm for signatures. Presented at the Dagstuhl Workshop on Cryptography (1997)
21. Patarin, J., Goubin, L., Courtois, N.: C_{-+}^* , and HM : variations around two schemes of T. Matsumoto and H. Imai. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 35–50. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49649-1_4
22. 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/10.1007/978-3-319-72565-9_18
23. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE public key cryptosystem by relinearization. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_2

24. Faugère, J., Din, M.S.E., Spaenlehauer, P.: Computing loci of rank defects of linear matrices using Gröbner bases and applications to cryptology. In: Koepf, W. (ed.) Proceedings of International Symposium on Symbolic and Algebraic Computation, ISSAC 2010, Munich, Germany, 25–28 July 2010, pp. 257–264. ACM (2010)
25. Bettale, L., Faugère, J., Perret, L.: Cryptanalysis of HFE, multi-HFE and variants for odd and even characteristic. *Des. Codes Cryptogr.* **69**, 1–52 (2013)
26. Ding, J., Hodges, T.J.: Inverting HFE systems is quasi-polynomial for all fields. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 724–742. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_41
27. Goubin, L., Courtois, N.T.: Cryptanalysis of the TTM cryptosystem. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 44–57. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_4
28. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I: the user language. *J. Symb. Comput.* **24**, 235–265 (1997). Computational algebra and number theory (London, 1993)
29. Yang, B.-Y., Chen, J.-M.: Theoretical analysis of XL over small fields. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 277–288. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27800-9_24
30. Bardet, M., Faugre, J., Salvy, B., Yang, B.: Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In: MEGA 2005 Eighth International Symposium On Effective Methods in Algebraic Geometry (2005)
31. Lange, T., Takagi, T. (eds.): PQCrypto 2017. LNCS, vol. 10346. Springer, Cham (2017). <https://doi.org/10.1007/978-3-319-59879-6>
32. Mosca, M. (ed.): PQCrypto 2014. LNCS, vol. 8772. Springer, Cham (2014). <https://doi.org/10.1007/978-3-319-11659-4>

Protocols



Post-Quantum Zero-Knowledge Proofs for Accumulators with Applications to Ring Signatures from Symmetric-Key Primitives

David Derler¹, Sebastian Ramacher^{1(✉)}, and Daniel Slamanig²

¹ IAIK, Graz University of Technology, Graz, Austria
{david.derler,sebastian.ramacher}@tugraz.at

² AIT Austrian Institute of Technology, Vienna, Austria
daniel.slamanig@ait.ac.at

Abstract. In this paper we address the construction of privacy-friendly cryptographic primitives for the post-quantum era and in particular accumulators with zero-knowledge membership proofs and ring signatures. This is an important topic as it helps to protect the privacy of users in online authentication or emerging technologies such as cryptocurrencies. Recently, we have seen first such constructions, mostly based on assumptions related to codes and lattices. We, however, ask whether it is possible to construct such primitives without relying on structured hardness assumptions, but solely based on symmetric-key primitives such as hash functions or block ciphers. This is interesting because the resistance of latter primitives to quantum attacks is quite well understood.

In doing so, we choose a modular approach and firstly construct an accumulator (with one-way domain) that allows to efficiently prove knowledge of (a pre-image of) an accumulated value in zero-knowledge. We, thereby, take care that our construction can be instantiated solely from symmetric-key primitives and that our proofs are of sublinear size. Latter is non trivial to achieve in the symmetric setting due to the absence of algebraic structures which are typically used in other settings to make these efficiency gains. Regarding efficient instantiations of our proof system, we rely on recent results for constructing efficient non-interactive zero-knowledge proofs for general circuits. Based on this building block, we then show how to construct logarithmic size ring signatures solely from symmetric-key primitives. As constructing more advanced primitives only from symmetric-key primitives is a very recent field, we discuss some interesting open problems and future research directions. Finally, we want to stress that our work also indirectly impacts other fields: for the first time it raises the requirement for collision resistant hash functions with particularly low AND count.

Keywords: Post-quantum cryptography

Privacy-preserving cryptography · Provable security · Accumulator
Zero-knowledge for circuits

1 Introduction

The design of cryptographic schemes that remain secure in the advent of powerful quantum computers has become an important topic in recent years. Although it is hard to predict when quantum computers will be powerful enough to break factoring and discrete logarithm based cryptosystems, it is important to start the transition to post-quantum cryptography early enough to eventually not end up in a rush. This is underpinned by the NIST post-quantum cryptography standardization project¹, which aims at identifying the next generation of public key encryption, key exchange and digital signature schemes basing their security on conjectured quantum hard problems. Apart from these fundamental schemes, there are many other valuable schemes which would nicely complement a post-quantum cryptographic toolbox. In this paper we are interested in privacy-friendly cryptographic primitives for the post-quantum era and in particular accumulators with zero-knowledge membership proofs and ring signatures. Such schemes help to protect the privacy of users, and significantly gained importance due to recent computing trends such as Cloud computing or the Internet of Things (IoT). Examples where privacy-enhancing protocols are already widely deployed today are remote attestation via direct anonymous attestation (DAA) [14] as used by the Trusted Platform Module (TPM)², privacy-friendly online authentication within Intel’s Enhanced Privacy ID (EPID) [15], or usage within emerging technologies such as cryptocurrencies to provide privacy of transactions.³

Let us now briefly discuss the primitives we construct in this paper. An accumulator scheme [10] allows to represent a finite set as a succinct value called the accumulator. For every element in the accumulated set, one can efficiently compute a so called witness to certify its membership in the accumulator. However, it should be computationally infeasible to find a witness for non-accumulated values. We are interested in accumulators supporting efficient zero-knowledge membership proofs. Ring signature schemes [40] allow a member of an ad-hoc group \mathcal{R} (the so called ring), defined by the member’s public keys, to anonymously sign a message on behalf of \mathcal{R} . Such a signature attests that some member of \mathcal{R} produced the signature, but the actual signer remains anonymous.

For ring signatures there is a known approach to construct them from accumulators and non-interactive zero-knowledge proof systems in the random oracle model. The main technical hurdle in the post-quantum setting is to find accumulators, and, more importantly, compatible proof systems under suitable assumptions. Only recently, Libert et al. in [34] showed that it is possible to instantiate this approach in the post-quantum setting and provided the first post-quantum accumulator from lattices. This combined with suitable non-interactive variants of Σ -protocols yields post-quantum ring signatures in the random oracle model (ROM). However, this does not give rise to a construction of ring signatures

¹ <https://csrc.nist.gov/groups/ST/post-quantum-crypto/>.

² <https://trustedcomputinggroup.org/tpm-library-specification/>.

³ <https://getmonero.org/resources/moneropedia/ringsignatures.html>.

from symmetric-key primitives such as hash functions or block ciphers, as we pursue in this paper. The main technical tools we use in our construction are recent results from zero-knowledge proof systems for general circuits [20, 30], and our techniques are inspired by recent approaches to construct post-quantum signature schemes based on these proof systems [20]. We note that there are also post-quantum ring signature candidates from problems related to codes [36] and multivariate cryptography [38]. However, they all have size linear in the number of ring members, whereas we are only interested in sublinear ones. Additionally, former schemes are proven secure in weaker security models.

Contribution. Our contributions are as follows:

- We present the first post-quantum accumulator (with one-way domain) together with efficient zero-knowledge proofs of (a pre-image of) an accumulated value, which solely relies on assumptions related to symmetric-key primitives. That is, we do not require any structured hardness assumptions. Our proofs are of sublinear size in the number of accumulated elements and can be instantiated in both, the ROM as well as the quantum accessible ROM (QROM). Besides being used as an important building block in this paper, such accumulators are of broader interest. In particular, such accumulators with efficient zero-knowledge membership proofs have many other applications beyond this work, e.g., membership revocation [6] or anonymous cash such as Zerocoin [37]. We also note that the only previous construction of post-quantum accumulators with efficient zero-knowledge membership proofs in [34] relies on hardness assumptions on lattices.
- We use our proposed accumulator to construct ring signatures of sub-linear size. Therefore, we prove an additional property—simulation-sound extractability—of the proof system (ZKB++ [20]) we are using. This then allows us to rigorously prove the security of our ring signature construction in the strongest model of security for ring signatures due to Bender et al. [11]. Consequently, we propose a construction of sublinear size ring signatures solely from symmetric-key primitives.
- We present a selection of symmetric-key primitives that can be used to instantiate our ring signature construction and evaluate the practicality of our approach. In particular, we present signature sizes for rings of various sizes when instantiating the one-way function and hash function using LowMC [3, 4]. Finally, we present some interesting directions for future research within this very recent domain.

2 Preliminaries

Notation. Let $x \stackrel{R}{\leftarrow} X$ denote the operation that picks an element uniformly at random from a finite set X and assigns it to x . We assume that all algorithms run in polynomial time and use $y \leftarrow A(x)$ to denote that y is assigned the output of the potentially probabilistic algorithm A on input x and fresh random coins. For algorithms representing adversaries we use calligraphic letters, e.g., \mathcal{A} . We

assume that every algorithm outputs a special symbol \perp on error. We write $\Pr[\Omega : \mathcal{E}]$ to denote the probability of an event \mathcal{E} over the probability space Ω . A function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}^+$ is called negligible if for all $c > 0$ there is a k_0 such that $\epsilon(k) < 1/k^c$ for all $k > k_0$. In the remainder of this paper, we use ϵ to denote such a negligible function. Finally, we define $[n] := \{1, \dots, n\}$.

2.1 Zero-Knowledge Proofs and Σ -Protocols

Σ -Protocols. Let $L \subseteq X$ be an NP-language with witness relation R so that $L = \{x \mid \exists w : R(x, w) = 1\}$. A Σ -protocol for language L is defined as follows.

Definition 1 (Σ -Protocol). A Σ -protocol for language L is an interactive three-move protocol between a PPT prover $P = (\text{Commit}, \text{Prove})$ and a PPT verifier $V = (\text{Challenge}, \text{Verify})$, where P makes the first move and transcripts are of the form $(\mathbf{a}, \mathbf{e}, \mathbf{z}) \in A \times E \times Z$, where \mathbf{a} is output by Commit , \mathbf{e} is output by Challenge and \mathbf{z} is output by Prove . Additionally, Σ protocols satisfy the following properties

Completeness. For all security parameters κ , and for all $(x, w) \in R$, it holds that

$$\Pr[(P(1^\kappa, x, w), V(1^\kappa, x)) = 1] = 1.$$

s -Special Soundness. There exists a PPT extractor E so that for all x , and for all sets of accepting transcripts $\{(\mathbf{a}, \mathbf{e}_i, \mathbf{z}_i)\}_{i \in [s]}$ with respect to x where $\forall i, j \in [s], i \neq j : \mathbf{e}_i \neq \mathbf{e}_j$, generated by any algorithm with polynomial runtime in κ , it holds that

$$\Pr[w \leftarrow E(1^\kappa, x, \{(\mathbf{a}, \mathbf{e}_i, \mathbf{z}_i)\}_{i \in [s]}) : (x, w) \in R] \geq 1 - \epsilon(\kappa).$$

Special Honest-Verifier Zero-Knowledge. There exists a PPT simulator S so that for every $x \in L$ and every challenge $\mathbf{e} \in E$, it holds that a transcript $(\mathbf{a}, \mathbf{e}, \mathbf{z})$, where $(\mathbf{a}, \mathbf{z}) \leftarrow S(1^\kappa, x, \mathbf{e})$ is computationally indistinguishable from a transcript resulting from an honest execution of the protocol.

The s -special soundness property gives an immediate bound for soundness: if no witness exists then (ignoring a negligible error) the prover can successfully answer at most to $(s - 1)/t$ challenges, where $t = |E|$ is the size of the challenge space. In case this value is too large, it is possible to reduce the soundness error using ℓ -fold parallel repetition of the Σ -protocol. Furthermore, it is also well known that one can easily express conjunctions and disjunctions of languages proven using Σ -protocols. For the formal details refer to [22, 23].

Non-interactive ZK Proof Systems. Now, we recall a standard definition of non-interactive zero-knowledge proof systems. Therefore, let L be an NP-language with witness relation R so that $L = \{x \mid \exists w : R(x, w) = 1\}$.

Definition 2 (Non-interactive Zero-Knowledge Proof System). A non-interactive proof system Π is a tuple of algorithms $(\text{Setup}, \text{Proof}, \text{Verify})$, defined as:

Setup(1^κ): This algorithm takes a security parameter κ as input, and outputs a common reference string crs .

Proof(crs, x, w): This algorithm takes a common reference string crs , a statement x , and a witness w as input, and outputs a proof π .

Verify(crs, x, π): This algorithm takes a common reference string crs , a statement x , and a proof π as input, and outputs a bit $b \in \{0, 1\}$.

We require the properties *completeness*, *adaptive zero-knowledge*, and *simulation-sound extractability*. Due to the lack of space, the definitions are presented in the full version.

The Fiat-Shamir Transform. The Fiat-Shamir transform [29] is a frequently used tool to convert Σ -protocols $\langle P, V \rangle$ to their non-interactive counterparts. Essentially, the transform removes the interaction between P and V by using a RO $H : A \times X \rightarrow E$ to obtain the challenge e .⁴ That is, one uses a PPT algorithm $\text{Challenge}'(1^\kappa, a, x)$ which obtains $e \leftarrow H(a, x)$ and returns e . Then, the prover can locally obtain the challenge e *after* computing the initial message a . Starting a verifier $V' = (\text{Challenge}', \text{Verify})$ on the same initial message a will then yield the same challenge e . More formally, we obtain the non-interactive PPT algorithms (P_H, V_H) indexed by the used RO:

$P_H(1^\kappa, x, w)$: Start P on $(1^\kappa, x, w)$, obtain the first message a , answer with $e \leftarrow H(a, x)$, and finally obtain z . Returns $\pi \leftarrow (a, z)$.

$V_H(1^\kappa, x, \pi)$: Parse π as (a, z) . Start V' on $(1^\kappa, x)$, send a as first message to V' . When V' outputs e , reply with z and output 1 if V' accepts and 0 otherwise.

One can obtain a non-interactive proof system satisfying the properties above by applying the Fiat-Shamir transform to any Σ -protocol where the min-entropy α of the commitment a sent in the first phase is so that $2^{-\alpha}$ is negligible in the security parameter κ and the challenge space E is exponentially large in the security parameter. Formally, $\text{Setup}(1^\kappa)$ fixes a RO $H : A \times X \rightarrow E$, sets $\text{crs} \leftarrow (1^\kappa, H)$ and returns crs . The algorithms Proof and Verify are defined as follows: $\text{Proof}(\text{crs}, x, w) := P_H(1^\kappa, x, w)$, $\text{Verify}(\text{crs}, x, \pi) := V_H(1^\kappa, x, \pi)$.

Signatures via Fiat-Shamir. The Fiat-Shamir (FS) transform can elegantly be used to convert (canonical) identification schemes into adaptively secure signature schemes. The basic idea is similar to above, but slightly differs regarding the challenge generation, i.e., one additionally includes the message upon generating the challenge. Note that in the context of the stronger variant of the FS transform we rely on, one can simply modify the language so that the statements additionally include the message to be signed. This is because our variant of the FS transform includes the statement upon challenge generation, which is why extending the statement by the message also implicitly means including the message in the challenge generation. We will not make this language change explicit in the following, but implicitly assume that the language is changed if a message is included as the last parameter of the statement to be proven.

⁴ This is a stronger variant of FS (cf. [12, 28]). The original weaker variant of the FS transform does not include the statement x in the challenge computation.

The Unruh Transform. Similar to FS, Unruh’s transform [41–43] allows one to construct NIZK proofs and signature schemes from Σ -protocols. In contrast to the FS transform, Unruh’s transform can be proven secure in the QROM (quantum random oracle model), strengthening the security guarantee against quantum adversaries. At a high level, Unruh’s transform works as follows: given Σ -protocol, the prover repeats the first phase of the Σ -protocol t times and for each of those runs produces responses for M randomly selected challenges. All those responses are permuted using a random permutation G . Querying the random oracle on all first rounds all permuted responses then determines the responses to publish for each round.

2.2 Efficient NIZK Proof Systems for General Circuits

ZKB++ [20], an optimized version of ZKBOO [30], is a proof system for zero-knowledge proofs over arbitrary circuits. ZKBOO and ZKB++ build on the MPC-in-the-head paradigm by Ishai et al. [33], which roughly works as follows. The prover simulates all parties of a multiparty computation protocol (MPC) implementing the joint evaluation of some function, say $y = \text{SHA-256}(x)$, and computes commitments to the states of all players. The verifier then randomly corrupts a subset of the players and checks whether those players did the computation correctly.

ZKBOO generalizes the idea of [33] by replacing MPC with circuit decompositions. There the idea is to decompose the circuit into three shares, where revealing the wire values of two shares does not leak any information about the wire values on the input of the circuit. The explicit formulas for circuit decomposition can be found in [30] for ZKBOO and in [20] for ZKB++. Multiplication gates induce some dependency between the individual shares which is why the wire values on the output of the multiplication gates needs to be stored in the transcripts. Hence, the transcripts grow linearly in the number of multiplication gates. Due to space limitations we do not include further details on ZKB++ and refer the reader to [20] for the details.

3 PQ Accumulators and ZK Membership Proofs

Our goal is to come up with an accumulator and associated efficient zero-knowledge membership proof system, which remains secure in the face of attacks by a quantum attacker. The first building block we, thus, require for our constructions are accumulators which can be proven secure under an assumption which is believed to resist attacks by a quantum computer. In this work our goal is to solely rely on unstructured assumptions, and thus resort to using Merkle-trees as accumulators. Merkle-trees were first used in the context of accumulators by Buldas et al. in [16], who called their primitive undeniable attestors. In the fashion of [27], we then extend the accumulator model to accumulators with one-way domain, i.e., accumulators where the accumulation domain coincides with the range of a one-way function so that one can accumulate images of the

one-way function. For the associated zero-knowledge membership proof system, we build up on recent progress in proving statements over general circuits as discussed in Sect. 2.2.

The main technical hurdle we face in this context is designing the statement to be proven with the proof system so that we can actually obtain proofs which are sublinear (in particular logarithmic) in the number of accumulated elements. Obtaining sublinear proofs is complicated mainly due to the absence of any underlying algebraic structure on the accumulator.

3.1 Formal Model

We rely on the formalization of accumulators by [24], which we slightly adapt to fit our requirement for a deterministic Eval algorithm. Based on this formalization we then restate the Merkle-tree accumulator (having a deterministic Eval algorithm) within this framework.

Definition 3 (Accumulator). *A static accumulator is a tuple of efficient algorithms (Gen, Eval, WitCreate, Verify) which are defined as follows:*

- Gen($1^\kappa, t$): This algorithm takes a security parameter κ and a parameter t . If $t \neq \infty$, then t is an upper bound on the number of elements to be accumulated. It returns a key pair $(\text{sk}_\Lambda, \text{pk}_\Lambda)$, where $\text{sk}_\Lambda = \emptyset$ if no trapdoor exists. We assume that the accumulator public key pk_Λ implicitly defines the accumulation domain \mathcal{D}_Λ .*
- Eval($(\text{sk}_\Lambda, \text{pk}_\Lambda), \mathcal{X}$): This deterministic algorithm takes a key pair $(\text{sk}_\Lambda, \text{pk}_\Lambda)$ and a set \mathcal{X} to be accumulated and returns an accumulator $\Lambda_{\mathcal{X}}$ together with some auxiliary information aux .*
- WitCreate($(\text{sk}_\Lambda, \text{pk}_\Lambda), \Lambda_{\mathcal{X}}, \text{aux}, x_i$): This algorithm takes a key pair $(\text{sk}_\Lambda, \text{pk}_\Lambda)$, an accumulator $\Lambda_{\mathcal{X}}$, auxiliary information aux and a value x_i . It returns \perp , if $x_i \notin \mathcal{X}$, and a witness wit_{x_i} for x_i otherwise.*
- Verify($\text{pk}_\Lambda, \Lambda_{\mathcal{X}}, \text{wit}_{x_i}, x_i$): This algorithm takes a public key pk_Λ , an accumulator $\Lambda_{\mathcal{X}}$, a witness wit_{x_i} and a value x_i . It returns 1 if wit_{x_i} is a witness for $x_i \in \mathcal{X}$ and 0 otherwise.*

We require accumulators to be correct and collision free. While we omit the straight forward correctness notion, we recall the collision freeness notion below, which requires that finding a witness for a non-accumulated value is hard.

Definition 4 (Collision Freeness). *A cryptographic accumulator is collision-free, if for all PPT adversaries \mathcal{A} there is a negligible function $\varepsilon(\cdot)$ such that:*

$$\Pr \left[\begin{array}{l} (\text{sk}_\Lambda, \text{pk}_\Lambda) \leftarrow \text{Gen}(1^\kappa, t), \\ (\text{wit}_{x_i}^*, x_i^*, \mathcal{X}^*) \leftarrow \mathcal{A}(\text{pk}_\Lambda) \end{array} : \begin{array}{l} \text{Verify}(\text{pk}_\Lambda, \Lambda^*, \text{wit}_{x_i}^*, x_i^*) = 1 \wedge \\ x_i^* \notin \mathcal{X}^* \end{array} \right] \leq \varepsilon(\kappa),$$

where $\Lambda^* \leftarrow \text{Eval}_{\text{pk}_\Lambda}((\text{sk}_\Lambda, \text{pk}_\Lambda), \mathcal{X}^*)$.

Gen($1^\kappa, t$): Fix a family of hash functions $\{H_k\}_{k \in \mathbb{K}^\kappa}$ with $H_k : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa \forall k \in \mathbb{K}^\kappa$. Choose $k \xleftarrow{R} \mathbb{K}^\kappa$ and return $(\text{sk}_\Lambda, \text{pk}_\Lambda) \leftarrow (\emptyset, H_k)$.

Eval($(\text{sk}_\Lambda, \text{pk}_\Lambda), \mathcal{X}$): Parse pk_Λ as H_k and \mathcal{X} as (x_0, \dots, x_{n-1}) .^a If $\nexists k \in \mathbb{N}$ so that $n = 2^k$ return \perp . Otherwise, let $\ell_{u,v}$ refer to the u -th leaf (the leftmost leaf is indexed by 0) in the v -th layer (the root is indexed by 0) of a perfect binary tree. Return $\Lambda_{\mathcal{X}} \leftarrow \ell_{0,0}$ and $\text{aux} \leftarrow ((\ell_{u,v})_{u \in [n/2^{k-v}]}_{v \in [k]})$, where

$$\ell_{u,v} \leftarrow \begin{cases} H_k(\ell_{2u,v+1} || \ell_{2u+1,v+1}) & \text{if } v < k, \text{ and} \\ H_k(x_i) & \text{if } v = k. \end{cases}$$

WitCreate($(\text{sk}_\Lambda, \text{pk}_\Lambda), \Lambda_{\mathcal{X}}, \text{aux}, x_i$): Parse aux as $((\ell_{u,v})_{u \in [n/2^{k-v}]}_{v \in [k]})$ and return wit_{x_i} where

$$\text{wit}_{x_i} \leftarrow (\ell_{\lfloor i/2^v \rfloor + \eta, k-v})_{0 \leq v \leq k}, \text{ where } \eta = \begin{cases} 1 & \text{if } \lfloor i/2^v \rfloor \pmod{2} = 0 \\ -1 & \text{otherwise.} \end{cases}$$

Verify($\text{pk}_\Lambda, \Lambda_{\mathcal{X}}, \text{wit}_{x_i}, x_i$): Parse pk_Λ as H_k , $\Lambda_{\mathcal{X}}$ as $\ell_{0,0}$, set $\ell_{i,k} \leftarrow H_k(x_i)$. Recursively check for all $0 < v < k$ whether the following holds and return 1 if so. Otherwise return 0.

$$\ell_{\lfloor i/2^{v+1} \rfloor, k-(v+1)} = \begin{cases} H_k(\ell_{\lfloor i/2^v \rfloor, k-v} || \ell_{\lfloor i/2^v \rfloor + 1, k-v}) & \text{if } \lfloor i/2^v \rfloor \pmod{2} = 0 \\ H_k(\ell_{\lfloor i/2^v \rfloor - 1, k-v} || \ell_{\lfloor i/2^v \rfloor, k-v}) & \text{otherwise.} \end{cases}$$

^a We assume without loss of generality that \mathcal{X} is an ordered sequence instead of a set.

Scheme 1. Merkle-tree accumulator.

3.2 The Accumulator

In Scheme 1, we cast the Merkle-tree accumulator in the framework of [24].

Then, we restate some well-known lemmas and sketch the respective proofs.

Lemma 1. *Scheme 1 is correct.*

The lemma above is easily verified by inspection. The proof is omitted.

Lemma 2. *If $\{H_k\}_{k \in \mathbb{K}^\kappa}$ is a family of collision resistant hash functions, the accumulator in Scheme 1 is collision free.*

Proof (Sketch). Upon setup, the reduction engages with a collision resistance challenger for the family of hash functions, obtains H_k , and completes the setup as in the original protocol. Now, one may observe that every collision in the accumulator output by the adversary implies that the reduction knows at least two colliding inputs for H_k , which upper bounds the probability of a collision in the accumulator by the collision probability of the hash function.

3.3 Accumulators with One-Way Domain

We now extend the definition of accumulators to ones with one-way domain following the definition of [27], but we adapt it to our notation.

Definition 5 (Accumulator with One-Way Domain). A collision-free accumulator with accumulation domain D_Λ and associated function family $\{f_\Lambda : \mathbb{I}_\Lambda \rightarrow D_\Lambda\}$ where $\text{Gen}(1^\kappa, t)$ also selects f_Λ is called an accumulator with one-way domain if

Efficient verification. There exists an efficient algorithm D that on input $(x, z) \in D_\Lambda \times \mathbb{I}_\Lambda$ returns 1 if and only if $f_\Lambda(z) = x$.

Efficient sampling. There exists a (probabilistic) algorithm W that on input 1^κ returns a pair $(x, z) \in D_\Lambda \times \mathbb{I}_\Lambda$ with $D(x, z) = 1$.

One-wayness. For all PPT adversaries \mathcal{A} there is a negligible function $\varepsilon(\cdot)$ such that:

$$\Pr [(x, z) \leftarrow W(1^\kappa), z^* \leftarrow \mathcal{A}(1^\kappa, x) : D(x, z) = 1] \leq \varepsilon(\kappa).$$

Note that when we set f_Λ to be the identity function, then we have a conventional accumulator.

3.4 Membership Proofs of Logarithmic Size

The main technical tool used by [27] to obtain zero-knowledge membership proofs of constant size is to exploit a property of the accumulator which is called quasi-commutativity. Clearly, such a property requires some underlying algebraic structure which we explicitly want to sacrifice in favor of being able to solely rely on assumptions related to symmetric-key primitives with relatively well understood post-quantum security. To this end we have to use a different technique. First observe that when naïvely proving that a non-revealed value is a member of our accumulator would amount to a disjunctive proof of knowledge over all members, which is at least of linear size. Therefore, this is not an option and we have to develop an alternative technique.

The Relation. Essentially our idea is to “emulate” some kind of commutativity within the order of the inputs to the hash function in each level by a disjunctive proof statement, i.e., we exploit the disjunction to hide where the path through the tree continues. The single statements in every level of the tree are then included in one big conjunction. The length of this statement is $\mathcal{O}(k) = \mathcal{O}(\log n)$. More formally we define a relation R on $\{0, 1\}^\kappa \times \{f_\Lambda\} \times \{H_k\} \times \mathbb{I}_\Lambda \times (\{0, 1\}^\kappa)^{2k}$ which—for a given non-revealed pre-image z —attests membership of the corresponding image $f_\Lambda(z)$ in the accumulator $\Lambda_{\mathcal{X}}$:

$$((\Lambda_{\mathcal{X}}, f_\Lambda, H_k), (z, (a_i)_{i \in [k]}, (b_i)_{i \in [k]})) \in R \iff (a_k = f_\Lambda(z) \vee b_k = f_\Lambda(z)) \wedge \bigwedge_{i=0}^{k-1} (a_i = H_k(a_{i+1} || b_{i+1}) \vee a_i = H_k(b_{i+1} || a_{i+1})),$$

where $\Lambda_{\mathcal{X}} = a_0$. In Fig. 1 we illustrate that the relation indeed works for arbitrary members of the accumulator without influencing the form of the statement or the witness. This illustrates that proving the statement in this way does not reveal

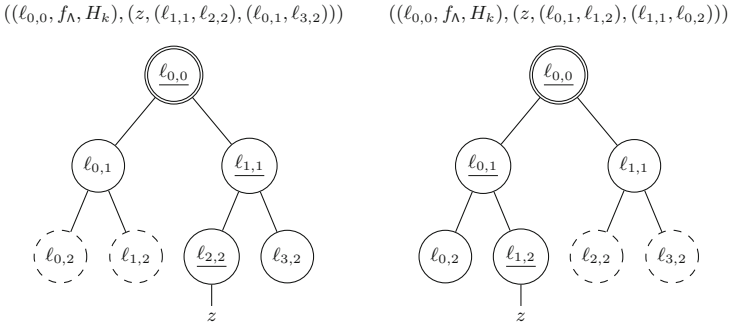


Fig. 1. Visualization of different paths in the Merkle-tree and the corresponding witness. The nodes on the path corresponding to a_0 , a_1 and a_2 are underlined.

any information on which path in the tree was taken. To see this, observe that at each level of the tree the relation covers both cases where a_i is either a left or right child. Given that, it is easy to verify that having a witness for relation R implies having a witness for the accumulator together with some (non-revealed) member.

Remark 1. In order to use relation R with the conventional accumulator in Scheme 1, we just have to set f_Λ to be the identity function (which yields $x = z$) and then set $a_k = H_k(z)$ and $b_k = H_k(z)$.

3.5 Converting Accumulator Witnesses

Now, the remaining piece to finally be able to plug in a witness $\text{wit}_{f_\Lambda(z)}$ for some accumulated value $f_\Lambda(z)$ with pre-image z into the relation R above is some efficient helper algorithm which rearranges the values z and $\text{wit}_{f_\Lambda(z)}$ so that they are compatible with the format required by R . Such an algorithm is easily implemented, which is why we only define the interface below.

$\text{Trans}(z, \text{wit}_{f_\Lambda(z)})$: Takes as input a value z as well as a witness $\text{wit}_{f_\Lambda(z)}$ and returns a witness of the form $(z, (a_i)_{i \in [k]}, (b_i)_{i \in [k]})$ for R .

Since Trans can be viewed as a permutation on the indexes it is easy to see that the function implemented by Trans is bijective and its inverse is easy to compute. We denote the computation of the inverse of the function implemented by Trans as $(z, \text{wit}_{f_\Lambda(z)}) \leftarrow \text{Trans}^{-1}(z, (a_i)_{i \in [n]}, (b_i)_{i \in [n]})$.

4 Logarithmic Size Ring Signatures

The two main lines of more recent work in the design of ring signatures target reducing the signature size or removing the requirement for random oracles (e.g., [13, 19, 26, 27, 31, 32, 35]). We, however, note that all these approaches

require assumptions that do not withstand a quantum computer. To the best of our knowledge, the first non-trivial post-quantum scheme (i.e., one that does not have linear size signatures) in the random oracle model is the lattice-based scheme recently proposed by Libert et al. [34]. We provide an alternative construction in the random oracle model with logarithmic sized signatures, but avoid lattice assumptions and only rely on symmetric-key primitives.

4.1 Formal Model

Below, we formally define ring signature schemes (adopting [11]).

Definition 6 (Ring Signature). A ring signature scheme RS is a tuple $RS = (\text{Setup}, \text{Gen}, \text{Sign}, \text{Verify})$ of PPT algorithms, which are defined as follows.

$\text{Setup}(1^\kappa)$: This algorithm takes as input a security parameter κ and outputs public parameters x_{pars} .

$\text{Gen}(\text{pp})$: This algorithm takes as input parameters pp and outputs a keypair (sk, pk) .

$\text{Sign}(\text{sk}_i, m, \mathcal{R})$: This algorithm takes as input a secret key sk_i , a message $m \in \mathcal{M}$ and a ring $\mathcal{R} = (\text{pk}_j)_{j \in [n]}$ of n public keys such that $\text{pk}_i \in \mathcal{R}$. It outputs a signature σ .

$\text{Verify}(m, \sigma, \mathcal{R})$: This algorithm takes as input a message $m \in \mathcal{M}$, a signature σ and a ring \mathcal{R} . It outputs a bit $b \in \{0, 1\}$.

A secure ring signature scheme needs to be correct, unforgeable, and anonymous. While we omit the obvious correctness definition, we subsequently provide formal definitions for the remaining properties following [11]. We note that Bender et al. in [11] have formalized multiple variants of these properties, where we always use the *strongest* one.

Unforgeability requires that without any secret key sk_i that corresponds to a public key $\text{pk}_i \in \mathcal{R}$, it is infeasible to produce valid signatures with respect to arbitrary such rings \mathcal{R} . Our unforgeability notion is the strongest notion defined in [11] and is there called *unforgeability w.r.t. insider corruption*.

Definition 7 (Unforgeability). A ring signature scheme provides unforgeability, if for all PPT adversaries \mathcal{A} , there exists a negligible function $\varepsilon(\cdot)$ such that it holds that

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{Setup}(1^\kappa), \\ \{(\text{sk}, \text{pk}) \leftarrow \text{Gen}(\text{pp})\}_{i \in [\text{poly}(\kappa)]}, \\ \mathcal{O} \leftarrow \{\text{Sig}(\cdot, \cdot, \cdot), \text{Key}(\cdot)\}, \\ (m^*, \sigma^*, \mathcal{R}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\{\text{pk}_i\}_{i \in [\text{poly}(\kappa)]}) \end{array} \quad : \quad \begin{array}{l} \text{Verify}(m^*, \sigma^*, \mathcal{R}^*) = 1 \wedge \\ (\cdot, m^*, \mathcal{R}^*) \notin \mathcal{Q}^{\text{Sign}} \wedge \\ \mathcal{R}^* \subseteq \{\text{pk}_i\}_{i \in [\text{poly}(\kappa)]} \setminus \mathcal{Q}^{\text{Key}} \end{array} \right] \leq \varepsilon(\kappa),$$

where $\text{Sig}(i, m, \mathcal{R}) := \text{Sign}(\text{sk}_i, m, \mathcal{R})$, Sig returns \perp if $\text{pk}_i \notin \mathcal{R} \vee i \notin [\text{poly}(\kappa)]$, and \mathcal{Q}^{Sig} records the queries to Sig . Furthermore, $\text{Key}(i)$ returns sk_i and \mathcal{Q}^{Key} records the queries to Key .

Anonymity requires that it is infeasible to tell which ring member produced a certain signature as long as there are at least two honest members in the ring. Our anonymity notion is the strongest notion defined in [11] and is there called *anonymity against full key exposure*.

Definition 8 (Anonymity). *A ring signature scheme provides anonymity, if for all PPT adversaries \mathcal{A} and for all polynomials $\text{poly}(\cdot)$, there exists a negligible function $\varepsilon(\cdot)$ such that it holds that*

$$\Pr \left[\begin{array}{l} \text{PP} \leftarrow \text{Setup}(1^\kappa), \\ \{(\text{sk}_i, \text{pk}_i) \leftarrow \text{Gen}(\text{PP})\}_{i \in [\text{poly}(\kappa)]}, \\ b \xleftarrow{\mathcal{R}} \{0, 1\}, \mathcal{O} \leftarrow \{\text{Sig}(\cdot, \cdot, \cdot)\}, \\ (m, j_0, j_1, \mathcal{R}, \text{st}) \leftarrow \mathcal{A}^\mathcal{O}(\{\text{pk}_i\}_{i \in [\text{poly}(\kappa)]}), \quad \{ \text{pk}_{j_i} \}_{i \in \{0,1\}} \subseteq \mathcal{R} \\ \sigma \leftarrow \text{Sign}(\text{sk}_{j_b}, m, \mathcal{R}), \\ b^* \leftarrow \mathcal{A}^\mathcal{O}(\text{st}, \sigma, \{\text{sk}_i\}_{i \in [\text{poly}(\kappa)]}) \end{array} \right] \leq 1/2 + \varepsilon(\kappa), \quad b = b^* \wedge$$

where $\text{Sig}(i, m, \mathcal{R}) := \text{Sign}(\text{sk}_i, m, \mathcal{R})$.

4.2 Generic Approaches to Design Ring Signatures

A folklore approach to design ring signatures in the random oracle model is to use the NP relation R_{RS} together with a one-way function μ , which defines the relation between secret and public keys:

$$(\mathcal{R}, \text{sk}) \in R_{\text{RS}} \iff \exists \text{pk}_i \in \mathcal{R}_{\text{RS}} : \text{pk}_i = \mu(\text{sk}),$$

and allows to demonstrate knowledge of a witness (a secret key) of one of the public keys in the ring \mathcal{R} . Usually, one then designs a Σ -protocol for relation R_{RS} and converts it into a signature scheme using the Fiat-Shamir heuristic.

Linear-size signatures. A frequently used instantiation of the above approach is instantiating the relation above by means of a disjunctive proof of knowledge [22]. Using this approach, one obtains ring signatures of linear size. It might be tempting to think that there is a lot of optimization potential for signature sizes in ring signatures. However, without additional assumptions about how the keys are provided to the verifier, signatures of linear size are already the best one can hope for: the verifier needs to get every public key in the ring to verify the signature.

Reducing signature size. However, to further reduce the signature size there is a nice trick which is based on the observation that in many practical scenarios the prospective ring members are already clear prior to the signature generation. Consequently, one can compactly encode all public keys in this ring within some suitable structure and compute the signatures with respect to this compact structure. This trick was first used by Dodis et al. [27]. Loosely their approach can be described as follows. They use a cryptographic accumulator with a one-way domain to accumulate the ring \mathcal{R} , a set of public keys being the output

of applying the one-way function μ to the respective secret key. This way they obtain a succinct representation of \mathcal{R} . Then, they use a proof system that allows to prove knowledge of a witness of one accumulated value (i.e., the public key) and knowledge of the pre-image thereof (i.e., the corresponding secret key). This proof can be turned into a signature using the Fiat-Shamir heuristic.

Depending on the size of the zero-knowledge membership proof this can yield sublinear (logarithmic or even constant size) signatures. Dodis et al. presented an instantiation of an accumulator together with the respective zero-knowledge proofs that yield constant size ring signatures based on the strong RSA assumption. Logarithmic size ring signatures under lattice assumptions are presented in [34].

4.3 Our Construction of Logarithmic Size Ring Signatures

Our construction basically follows the approach discussed above to reduce signature size. However, in contrast to Dodis et al., besides targeting the post-quantum setting, we (1) *do not* require a trusted setup⁵, and (2) cannot rely on accumulators with one-way domain which provide *quasi-commutativity*. Latter is too restricting and not compatible with the setting in which we work. In particular, it excludes Merkle-tree accumulators, which is why we chose to rely on a more generic formalization of accumulators (cf. Sect. 3). Like Dodis et al., we assume that in practical situations rings often stay the same for a long period of time (e.g., some popular rings are used very often by various members of the ring), or have an implicit short description. Consequently, we measure the signature size as that of the actual signature, i.e., the information one requires *in addition* to the group description. We want to stress once again that when counting the description of the ring as part of the signature, every secure ring signature schemes needs to have signature sizes which are at least linear in the size of the ring.

For the ease of presentation let us fix one such popular ring \mathcal{R} identified by the corresponding accumulator $\Lambda_{\mathcal{R}}$ and we assume that $|\mathcal{R}| = 2^t$ for some $t \in \mathbb{N}$.⁶ We present our construction as Scheme 2.

Remark 2. Note that in Scheme 2 crs is not a common reference string (CRS) that needs to be honestly computed by a trusted third party. We simply stick with the notion including a CRS for formal reasons, i.e., to allow the abstract notion of NIZKs, but as we exclusively use NIZK from Σ -protocols, we do not require a trusted setup and crs is just a description of the hash function which can be globally fixed, e.g., to SHA-256 or SHA-3. Recall, within Fiat-Shamir $\Pi.\text{Setup}(1^\kappa)$ fixes a RO $H : A \times X \rightarrow E$, sets $\text{crs} \leftarrow (1^\kappa, H)$ and returns crs .

Remark 3. A trusted setup in context of ring signatures is actually problematic, as it assumes that some mutually trusted party honestly executes the setup.

⁵ A trusted setup somehow undermines the idea behind ring signatures.

⁶ If this is not the case, one can always add dummy keys to the ring to satisfy this condition.

Setup(1^κ): Let Λ be the accumulator with one-way domain based on Scheme 1, run $(\text{sk}_\Lambda, \text{pk}_\Lambda) \leftarrow \Lambda.\text{Gen}(1^\kappa, t)$ (note that $\text{sk}_\Lambda = \emptyset$). Run $\text{crs} \leftarrow \Pi.\text{Setup}(1^\kappa)$ and return $\text{PP} \leftarrow (\text{pk}_\Lambda, \text{crs}) = ((H_k, f_\Lambda), (1^\kappa, H))$.

KeyGen(PP): Parse PP as $((H_k, f_\Lambda), \text{crs})$, run $(x, z) \leftarrow f_\Lambda.W(1^\kappa)$, and set $\text{pk} \leftarrow (\text{PP}, x)$, $\text{sk} \leftarrow (\text{pk}, z)$. Return (sk, pk) .

Sign($\text{sk}_i, m, \mathcal{R}$): Parse sk_i as $((((H_k, f_\Lambda), \text{crs}), x_i), z_i)$ and \mathcal{R} as $(\text{pk}_1, \dots, \text{pk}_t) = ((\cdot, x_1), \dots, (\cdot, x_t))$. Let $\mathcal{X} = (x_1, \dots, x_t)$, run $(\Lambda_{\mathcal{X}}, \text{aux}) \leftarrow \Lambda.\text{Eval}(\cdot, \text{pk}_\Lambda, \mathcal{X})$ and $\text{wit}_{f_\Lambda(z_i)} \leftarrow \Lambda.\text{WitCreate}(\cdot, \text{pk}_\Lambda, \Lambda_{\mathcal{X}}, \text{aux}, f_\Lambda(z_i))$. Obtain $(z_i, (a_j)_{j \in [t]}, (b_j)_{j \in [t]}) \leftarrow \text{Trans}(z_i, \text{wit}_{f_\Lambda(z_i)})$, and return the signature $\sigma \leftarrow (\pi, \Lambda_{\mathcal{X}})$, where

$$\pi \leftarrow \Pi.\text{Proof}(\text{crs}, (\Lambda_{\mathcal{X}}, f_\Lambda, H_k), (z_i, (a_j)_{j \in [t]}, (b_j)_{j \in [t]})).$$

Verify(m, σ, \mathcal{R}): Parse σ as $(\pi, \Lambda_{\mathcal{X}})$ and \mathcal{R} as $(\text{pk}_1, \dots, \text{pk}_t) = (((H_k, f_\Lambda), \text{crs}), x_1), \dots, (\cdot, x_t)$. Let $\mathcal{X} = (x_1, \dots, x_t)$, and compute

$$(\Lambda'_{\mathcal{X}}, \text{aux}') \leftarrow \Lambda.\text{Eval}(\cdot, \text{pk}_\Lambda, \mathcal{X}).$$

If $\Lambda'_{\mathcal{X}} \neq \Lambda_{\mathcal{X}}$ return 0. Otherwise return $\Pi.\text{Verify}(\text{crs}, (\Lambda_{\mathcal{X}}, f_\Lambda, H_k), \pi)$.

Scheme 2. Construction of logarithmic size RS.

For instance, in case of the strong RSA accumulator [7, 18] as used within [27], the party running the Gen algorithm of the accumulator can arbitrarily cheat. This can easily be done by keeping the accumulator secret (a trapdoor) instead of discarding it. Using this information, a dishonest setup allows to insert and delete arbitrary elements into and from the accumulator without changing the accumulator value. In context of ring signatures one thus can arbitrarily modify existing rings used within signatures, which could lead to modification of rings to just include public keys into the ring so that for every member of the ring the sole fact to know that one of these persons produced a signature already leads to severe consequences. *We stress that in our case there is no trusted setup. In particular, there is no accumulator secret and the public parameters are just descriptions of hash functions and a OWF.*

Now, we argue that our ring signature presented in Scheme 2 represents a secure ring signature scheme, where we omit correctness which is straightforward to verify.

Theorem 1. *If Λ is a collision free accumulator with one-way domain with respect to f_Λ and Π is a simulation-sound extractable non-interactive proof system, then the ring signature scheme in Scheme 2 is unforgeable.*

Proof. We prove unforgeability using a sequence of games.

Game 0: The original unforgeability game.

Game 1: As Game 0, but we modify Gen to setup (crs, τ) using \mathcal{S}_1 and henceforth simulate all proofs in Sign without a witness using τ .

Transition - Game 0 \rightarrow Game 1: A distinguisher between Game 0 and Game 1 is a zero-knowledge distinguisher for Π , i.e., $|\Pr[S_0] - \Pr[S_1]| \leq \varepsilon_{\text{zk}}(\kappa)$.

Game 2: As Game 1, but we further modify Gen to setup (crs, τ, ξ) using \mathcal{E}_1 and store ξ .

Transition - Game 1 \rightarrow Game 2: By simulation-sound extractability, this change is only conceptual, i.e., $\Pr[S_1] = \Pr[S_2]$.

Game 3: As Game 2, but for the forgery $(m^*, \sigma^*, \mathcal{R}^*)$ output by the adversary we parse σ^* as $(\pi, \Lambda_{\mathcal{X}})$ and obtain $(z_i, (a_i)_{i \in [k]}, (b_i)_{i \in [k]}) \leftarrow \mathcal{E}_2(\text{crs}, \xi, (\Lambda_{\mathcal{X}}, f_{\Lambda}, H_k), \pi)$. If the extractor fails, we abort.

Transition - Game 2 \rightarrow Game 3: Game 2 and Game 3 proceed identically, unless we abort. The probability for the abort event to happen is upper bounded by $\varepsilon_{\text{ext}}(\kappa)$ which is why we can conclude that $|\Pr[S_3] - \Pr[S_2]| \leq \varepsilon_{\text{ext}}(\kappa)$.

Game 4: As Game 3, but we abort if we have extracted $(z_i, (a_i)_{i \in [n]}, (b_i)_{i \in [n]})$ so that $(\cdot, \text{wit}_{f_{\Lambda}(z_i)}) \leftarrow \text{Trans}^{-1}(z_i, (a_i)_{i \in [n]}, (b_i)_{i \in [n]})$ is a valid witness for some $f_{\Lambda}(z_i)$ which was never accumulated.

Transition - Game 3 \rightarrow Game 4: If we abort in Game 4, we have a collision for the accumulator. That is $|\Pr[S_3] - \Pr[S_4]| \leq \varepsilon_{\text{cf}}(\kappa)$.

Game 5: As Game 4, but we guess the index i^* the adversary will attack beforehand, and abort if our guess is wrong.

Transition - Game 4 \rightarrow Game 5: The success probability in Game 4 is the same as in Game 5, unless our guess is wrong, i.e., $\Pr[S_5] = 1/\text{poly}(\kappa) \cdot \Pr[S_4]$.

Game 6: As Game 5, but instead of honestly generating the keypair for user i^* , we engage with a challenger of a OWF to obtain x_{i^*} and include it in pk_{i^*} accordingly. We set $\text{sk}_{i^*} \leftarrow \emptyset$.

Transition - Game 5 \rightarrow Game 6: This change is conceptual, i.e., $\Pr[S_5] = \Pr[S_6]$.

In the last game, we have an adversary against the OWF, i.e., $\Pr[S_6] \leq \varepsilon_{\text{owf}}(\kappa)$. All in all, we have that $\Pr[S_0] \leq \text{poly}(\kappa) \cdot \varepsilon_{\text{owf}}(\kappa) + \varepsilon_{\text{zk}}(\kappa) + \varepsilon_{\text{ext}}(\kappa) + \varepsilon_{\text{cf}}(\kappa)$.

Theorem 2. *If Π is a zero-knowledge non-interactive proof system, then the ring signature scheme in Scheme 2 is anonymous.*

Proof. We prove anonymity using a sequence of games.

Game 0: The original anonymity game.

Game 1: As Game 0, but we modify Gen to setup (crs, τ) using \mathcal{S}_1 and henceforth simulate all proofs in Sign without a witness using τ .

Transition - Game 0 \rightarrow Game 1: A distinguisher between Game 0 and Game 1 is a zero-knowledge distinguisher for Π , i.e., $|\Pr[S_0] - \Pr[S_1]| \leq \varepsilon_{\text{zk}}(\kappa)$.

In Game 1 the simulation is independent of b , meaning that $\Pr[S_1] = 1/2$. Thus, we have $\Pr[S_0] \leq 1/2 + \varepsilon_{\text{zk}}(\kappa)$, which concludes the proof. \square

5 Implementation Aspects and Evaluation

In this section we discuss some implementation aspects regarding instantiating our ring signature scheme. Moreover, we evaluate the efficiency of a concrete

instantiation. Since we require simulation-sound extractable NIZK proof systems, we confirm that the Fiat-Shamir (resp. Unruh) transformed version of ZKB++ represents a suitable proof system in the ROM (resp. QROM). We again want to note that we were not able to include the ZKB++ construction due to space limitations, but refer the reader to [20] for the details.

5.1 Simulation-Sound Extractability of ZKB++

To instantiate our ring signature scheme using ZKB++, we first need to confirm that the NIZK proof system obtained by applying the Fiat-Shamir/Unruh transform to ZKB++ is in fact simulation-sound extractable. For the Unruh-transformed proof system, this was already shown in [20, Theorem 2] in the QROM, which is why we only focus on the Fiat-Shamir version. We base our argumentation upon the argumentation in [28]. What we have to do is to show that the FS transformed ZKB++ is zero-knowledge and provides quasi-unique responses in the ROM. We do so by proving two lemmas. Combining those lemmas with [28, Theorems 2 and 3] then yields simulation-sound extractability as a corollary.

Lemma 3. *Let Q_H be the number of queries to the random oracle H , Q_S be the overall queries to the simulator, and let the commitments be instantiated via a RO H' with output space $\{0, 1\}^\rho$ and the committed values having min entropy ν . Then the probability $\epsilon(\kappa)$ for all PPT adversaries \mathcal{A} to break zero-knowledge of κ parallel executions of the FS transformed ZKB++ is bounded by $\epsilon(\kappa) \leq s/2^\nu + (Q_S \cdot Q_H)/2^{3 \cdot \rho}$.*

The lemma above was already proven for ZKBOO in [25]. For ZKB++ the argumentation is the same. We restate the proof below for completeness.

Proof. We bound the probability of any PPT adversary \mathcal{A} to win the zero-knowledge game by showing that the simulation of the proof oracle is statistically close to the real proof oracle. For our proof let the environment maintain a list H where all entries are initially set to \perp .

Game 0: The zero-knowledge game where the proofs are honestly computed, and the ROs are simulated honestly.

Game 1: As Game 0, but whenever the adversary requests a proof for some tuple (x, w) we choose $e \xleftarrow{R} \{0, 1, 2\}^\kappa$ before computing a and z . If $H[(a, x)] \neq \perp$ we abort and call that event E . Otherwise, we set $H[(a, x)] \leftarrow e$.

Transition - Game 0 \rightarrow Game 1: Both games proceed identically unless E happens. The message a includes 3 RO commitments with respect to H' , i.e., the min-entropy is lower bounded by $3 \cdot \rho$. We have $|\Pr[S_0] - \Pr[S_1]| \leq (Q_S \cdot Q_H)/2^{3 \cdot \rho}$.

Game 2: As Game 1, but we compute the commitments in a so that the ones which will never be opened according to e contain random values.

Transition - Game 1 \rightarrow Game 2: The statistical difference between Game 1 and Game 2 can be upper bounded by $|\Pr[S_1] - \Pr[S_2]| \leq \kappa \cdot 1/2^\nu$ (for compactness we collapsed the s game changes into a single game).

Game 3: As Game 2, but we use the HVZK simulator to obtain (a, e, z) .

Transition - Game 2 \rightarrow Game 3: This change is conceptual, i.e., $\Pr[S_2] = \Pr[S_3]$.

In Game 0, we sample from the first distribution of the zero-knowledge game, whereas we sample from the second one in Game 3; the distinguishing bounds shown above conclude the proof. \square

Lemma 4. *Let the commitments be instantiated via a RO H' with output space $\{0, 1\}^p$ and let $Q_{H'}$ be the number of queries to H' , then the probability to break quasi-unique responses is bounded by $Q_{H'}^2/2^p$.*

Proof. To break quasi-unique responses, the adversary would need to come up with two valid proofs (a, e, z) and (a, e, z') . The last message z (resp z') only contains openings to commitments, meaning that breaking quasi unique responses implies finding a collision for at least one of the commitments. The probability for this to happen is upper bounded by $Q_{H'}^2/2^p$ which concludes the proof. \square

Combining Lemmas 3 and 4 with [28, Theorems 2 and 3] yields the following corollary.

Corollary 1. *The FS transformed ZKB++ is simulation-sound extractable.*

5.2 Selection of Symmetric-Key Primitives

When instantiating our ring signature scheme using ZKB++, the selection of the underlying primitives is of importance for the actual signature sizes as well as the overall performance. As ZKB++'s proof size depends on the number of multiplication gates and the size of the operands, we require a OWF and a collision-resistant hash function with a representation as circuit, where the product of the multiplicative complexity and the number of bits required to store field elements is minimal. Note that for the OWF we can observe that, when instantiating it with a block cipher, only one plaintext-ciphertext pair per key is visible to an adversary. Hence, we have the same requirements as in [20], which is why we also choose LowMC [3, 4] with a reduced data complexity to build the OWF. For the selection of the collision-resistant hash function we are presented with different options:

Standardized Hash Functions. SHA-256 or SHA-3 are the obvious choices for collision resistant hash functions. SHA-256's compression function requires around 25000 multiplication gates [9] and SHA-3's permutation even more with around 38400 gates [39].

Sponge Construction with Low Multiplicative Complexity Ciphers.

Using a block cipher with small multiplicative complexity as permutation in a sponge construction, e.g., using LowMC or MiMC [2], enables the construction of hash functions with similar security guarantees as SHA-256 and SHA-3, but with a significantly reduced multiplicative complexity. Using the numbers from [2], MiMCHash-256 requires 1293 multiplications with a field size of 1025 bits.

LowMCHash-256 only requires a 1 bit binary field and 3540 AND gates⁷. Thus, a hash based on LowMC is a better candidate for our use case.

Finally we present signature sizes when instantiating our ring signature scheme with LowMC for both OWF and the hash function. Table 1 presents signature size estimations for some choices of ring sizes and aiming at a 128 bit post-quantum security level and we compute them using the formulas from [20]. For the Fiat-Shamir-transformed proof system the involved proofs have a maximal size of $t \cdot (c + 2s + \log_2(3) + \ell \cdot m + 2i)$ bits and $t \cdot (c + 3s + \log_2(3) + 2\ell \cdot m + 2i)$ bits for the Unruh-transformed proofs, where t is the number of repetitions, c the size of the commitments, i the size of the input to the circuit, ℓ the size of the underlying field, m the number of AND gates, and s the size of the seeds used to generate the random tapes. We use ZKB++ as instantiated in [20] and give the numbers for both the Fiat-Shamir and Unruh transformed proof system.

Table 1. Signature sizes at the 128 bit post-quantum security level.

Ring size	$ \sigma $ (FS/ROM)	$ \sigma $ (Unruh/QROM)
2^k	$1335900 + 3213168 \cdot k$ bits	$2059476 + 4763688 \cdot k$ bits
2^5	2125 KB	3159 KB
2^{10}	4086 KB	6067 KB
2^{20}	8008 KB	11882 KB

We note that Ligerio [5], a recent NIZK proof system for general circuits, offers proofs logarithmic in the number of multiplication gates in the prime field case respectively in the number of AND and XOR gates in the case of binary fields, which would allow us to reduce the signature size significantly. However, to the best of our knowledge, it is unclear whether Ligerio provides simulation-sound extractability.

6 Conclusions

In this work we made some important steps towards establishing privacy-enhancing primitives which are solely built from symmetric-key primitives and therefore do not require any structured hardness assumptions. In our work, we followed a modular concept and first introduced a post-quantum accumulator with efficient zero-knowledge membership proofs of sublinear size. Besides the applications to logarithmic size ring signatures as we presented in this paper, we believe that our post-quantum accumulator construction with zero-knowledge proofs may well have broader impact in the construction of other (privacy-enhancing) protocols in the post-quantum setting.

⁷ Numbers updated according to a personal discussion with Christian Rechberger.

Open Questions. In addition, we believe that our work also opens up quite some possibilities for further research.

First, in the context of privacy-enhancing protocols, it would be interesting to investigate how to extend our methods to obtain group signatures [21], i.e., anonymous signatures that provide the possibility to re-identify anonymous signers by a dedicated party. We note that Dodis et al. [27] informally discuss that when adding ID escrow functionality to their ring signature scheme yields group signatures. Basically, the lattice-based construction of Libert et al. [34] can be considered as an instantiation of the former paradigm. The problem is that this paradigm requires IND-CCA2 secure public-key encryption, which does not exist given our constraints. In addition, it is well known [1, 17] that group signatures in the static model by Bellare et al. in [8] imply public-key encryption. This means that the best one could hope for would be a construction being secure in a weakened version of the Bellare et al. model. Work in this direction was earlier pursued by Camenisch and Groth [17], who showed how to construct group signature schemes in a weaker model from one-way functions and non-interactive zero-knowledge arguments. The question which remains open in our context is whether one can find instantiations without the requirement for structured hardness assumptions and providing the practical efficiency one would hope for, i.e., ideally instantiations which just require to prove statements with respect to a few evaluations of a block cipher.

Second, in the context of symmetric-key primitives, one may observe that—despite the recent trend to construct symmetric-key primitives with particularly low AND count—there is no practical application so far which would require collision resistant hash functions with particularly low AND count. Since our accumulator construction relies on collision resistant hash functions, our work may well also open up new fields of research in the symmetric-key community.

Acknowledgments. The authors have been supported by EU H2020 Project PRISMACLOUD, grant agreement n°644962. We thank Christian Rechberger for discussions on the choice of symmetric-key primitives, especially regarding the instantiation of hash functions using LowMC, as well as for providing us with updated LowMC instances.

References

1. Abdalla, M., Warinschi, B.: On the minimal assumptions of group signature schemes. In: Lopez, J., Qing, S., Okamoto, E. (eds.) ICICS 2004. LNCS, vol. 3269, pp. 1–13. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30191-2_1
2. Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 191–219. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_7
3. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 430–454. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_17


4. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. IACR Cryptology ePrint Archive 2016, 687 (2016)
5. Ames, S., Hazay, C., Ishai, Y., Venkatasubramanian, M.: Liger: lightweight sub-linear arguments without a trusted setup. In: CCS (2017)
6. Baldimtsi, F., Camenisch, J., Dubovitskaya, M., Lysyanskaya, A., Reyzin, L., Samelin, K., Yakoubov, S.: Accumulators with applications to anonymity-preserving revocation. In: IEEE EuroS&P 2017 (2017)
7. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_33
8. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_38
9. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: decentralized anonymous payments from bitcoin. In: IEEE SP (2014)
10. Benaloh, J., de Mare, M.: One-way accumulators: a decentralized alternative to digital signatures. In: Hellese, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_24
11. Bender, A., Katz, J., Morselli, R.: Ring signatures: stronger definitions, and constructions without random oracles. *J. Cryptol.* **22**(1), 114–138 (2009)
12. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: pitfalls of the fiat-shamir heuristic and applications to helios. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 626–643. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_38
13. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J., Petit, C.: Short accountable ring signatures based on DDH. In: Pernul, G., Ryan, P.Y.A., Weippl, E. (eds.) ESORICS 2015. LNCS, vol. 9326, pp. 243–265. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24174-6_13
14. Brickell, E.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In: CCS (2004)
15. Brickell, E., Li, J.: Enhanced privacy ID: a direct anonymous attestation scheme with enhanced revocation capabilities. In: WPES (2007)
16. Buldas, A., Laud, P., Lipmaa, H.: Accountable certificate management using undeniable attestations. In: CCS (2000)
17. Camenisch, J., Groth, J.: Group signatures: better efficiency and new theoretical aspects. In: SCN (2004)
18. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_5
19. Chandran, N., Groth, J., Sahai, A.: Ring signatures of sub-linear size without random oracles. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 423–434. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73420-8_38
20. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: CCS (2017)

21. Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_22
22. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_19
23. Damgård, I.: On Σ -protocols (2010). <http://www.cs.au.dk/~ivan/Sigma.pdf>
24. Derler, D., Hanser, C., Slamanig, D.: Revisiting cryptographic accumulators, additional properties and relations to other primitives. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 127–144. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16715-2_7
25. Derler, D., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D.: Digital signatures from symmetric-key primitives. IACR Cryptology ePrint Archive 2016, 1085 (2016)
26. Derler, D., Slamanig, D.: Key-homomorphic signatures and applications to multiparty signatures and non-interactive zero-knowledge. IACR Cryptology ePrint Archive 2016, 792 (2016)
27. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in *Ad Hoc* groups. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 609–626. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_36
28. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the fiat-shamir transform. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 60–79. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34931-7_5
29. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
30. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: faster zero-knowledge for boolean circuits. In: USENIX Security (2016)
31. González, A.: A ring signature of size $\theta(\sqrt[3]{n})$ without random oracles. Cryptology ePrint Archive, Report 2017/905 (2017)
32. Groth, J., Kohlweiss, M.: One-out-of-many proofs: or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 253–280. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_9
33. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge proofs from secure multiparty computation. SIAM J. Comput. **39**(3), 1121–1152 (2009)
34. Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: logarithmic-size ring signatures and group signatures without trapdoors. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 1–31. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_1
35. Malavolta, G., Schröder, D.: Efficient ring signatures in the standard model. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 128–157. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_5
36. Melchor, C.A., Cayrel, P.-L., Gaborit, P.: A new efficient threshold ring signature scheme based on coding theory. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 1–16. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88403-3_1

37. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: anonymous distributed e-cash from bitcoin. In: IEEE S&P (2013)
38. Mohamed, M.S.E., Petzoldt, A.: RingRainbow – an efficient multivariate ring signature scheme. In: Joye, M., Nitaj, A. (eds.) AFRICACRYPT 2017. LNCS, vol. 10239, pp. 3–20. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57339-7_1
39. NIST: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology (NIST), FIPS PUB 202, U.S. Department of Commerce (2015)
40. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_32
41. Unruh, D.: Quantum proofs of knowledge. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 135–152. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_10
42. Unruh, D.: Non-interactive zero-knowledge proofs in the quantum random oracle model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 755–784. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_25
43. Unruh, D.: Computationally binding quantum commitments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 497–527. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_18



G-Merkle: A Hash-Based Group Signature Scheme from Standard Assumptions

Rachid El Bansarkhani¹  and Rafael Misoczki²

¹ Technische Universität Darmstadt, 64289 Darmstadt, Germany
elbansarkhani@cdc.informatik.tu-darmstadt.de

² Intel Corporation, Santa Clara, USA
rafael.misoczki@intel.com

Abstract. Hash-based signature schemes are the most promising cryptosystem candidates in a post-quantum world, but offer little structure to enable more sophisticated constructions such as group signatures. Group signatures allow a group member to anonymously sign messages on behalf of the whole group (as needed for anonymous remote attestation). In this work, we introduce G-Merkle, the first (stateful) hash-based group signature scheme. Our proposal relies on minimal assumptions, namely the existence of one-way functions, and offers performance equivalent to the Merkle single-signer setting. The public key size (as small as in the single-signer setting) outperforms all other post-quantum group signatures. Moreover, for N group members issuing at most B signatures each, the size of a hash-based group signature is just as large as a Merkle signature with a tree composed by $N \cdot B$ leaf nodes. This directly translates into fast signing and verification engines. Different from lattice-based counterparts, our construction does not require any random oracle. Note that due to the randomized structure of our Merkle tree, the signature authentication paths are pre-stored or deduced from a public tree, which seems a requirement hard to circumvent. To conclude, we present implementation results to demonstrate the practicality of our proposal.

Keywords: Group signatures · Hash-based signatures
Post-quantum cryptography

1 Introduction

Post-quantum cryptography is attracting increasing attention since the recent announcements by NIST [oSN16], NSA [NSA15] and the PQCRYPTO project [PQC16] that endorse the migration from classical to post-quantum schemes. Hash-based signatures (HBS) are considered good candidates as they offer good security and performance guarantees. They are considered quantum resistant, while widely-deployed public key cryptosystems are susceptible to polynomial-time quantum attacks [Sho94], and rely only on minimal assumptions, namely

certain well-studied security notions related to hash functions. Note that any signature scheme (classical or post-quantum) with appended message also relies on the security of hashing (used to map arbitrary length message into a fixed length digest), plus some other (likely less studied) assumptions.

Another strength of HBS refers to their practical performance. As opposed to conventional schemes, where expensive computations are required, HBS only require hash computations, an operation with performance akin to symmetric key cryptography rather than public-key cryptosystems. Given their high efficiency and tight security, HBS can be seen as one of the fewest post-quantum cryptographic alternatives that can immediately replace conventional cryptosystems (although stateful schemes may require extra caution related to state management [MKF+16]). However, the same simplicity that leads to high efficiency and tight security also imposes limitations to build more sophisticated constructions such as group signatures.

Group signatures allow any member of a group to anonymously sign messages on behalf of a group. This is accomplished by a unique group public key that is the same for all group members. A group manager can break the anonymity of any group signature by means of a master key and thus determine the respective issuer (*traceability*). Note that no other entity other than the group manager is able to gather information or trace a signature back to any group member (*anonymity*). Group signature schemes have great applicability in real-world, such as in remote attestation protocols, traffic management, e-commerce, e-cash, e-voting and e-auction, just to name a few examples.

1.1 Related Work

The first group signature scheme has been introduced in [Cv91]. Subsequently, it has been improved in [ACJT00]. The notion of full-anonymity and full-traceability can be traced back to the security model proposed in [BMW03], which allows for even stronger security properties, once these notions are established in a scheme. Since then, a great deal of practical constructions based on classical assumptions have been proposed. Those schemes can be classified into random oracle based constructions [ACJT00, CL02, CL04, DP06] and standard model variants [BMW03, BSZ05, BW06, BW07, Gro07]. All of these constructions are based on Groth-Sahai's non-interactive proof systems (NIZK) for a great deal of languages. In [BL09], Brickell and Li introduced EPID with advanced properties such as signature-based and private-key based revocation. There are only a few (secure) constructions based on computational problems that are believed to be quantum resistant. Such schemes are mainly based on lattice-based hardness assumptions [GKV10, LLS13, LLNW14, NZZ15, LNW15] or on code-based scheme [ELL+15] that relies on additional (non-usual) assumptions. However, all of those constructions require expensive non-interactive zero-knowledge arguments for specific languages such as [MV03].

Previously, it seemed hard to construct group signature schemes out of hash functions as they offer little structure to exploit. In fact, there exists little

literature on special property signature schemes from hash functions. One example is the forward secure signature scheme (also proxy- and key-insulated signature schemes) from one-way functions by Buchmann et al. [BDH11]. Recently, it has also been shown that NIZK proofs [GMO16] can be built out of hash functions and techniques from multi-party computation. This opens new directions as NIZK proof systems serve as a common tool to realize advanced cryptographic constructions. However, to the best of our knowledge, no hash-based group signature scheme has ever been proposed in the literature.

1.2 Our Contributions

In this work, we propose the first (stateful) hash-based group signature scheme. Our proposal has many advantages over other group signature schemes such as:

- It is very simple as it is solely built from a regular Merkle tree based signature scheme in combination with a secure block cipher or pseudo random function. The latter can be built out of one-way functions by the construction of Luby and Rackoff [LR86], hence allowing for a construction based on minimal assumptions in the standard model. This answers the open question raised in [LNW15].
- We do not require expensive non-interactive zero knowledge proofs (e.g. via the Fiat-Shamir Transform) as used in other group signature schemes in order to prove possession of a secret. As a result, no random oracle instantiation is needed.
- It is post-quantum secure with small key and signature sizes, outperforming all other post-quantum group signature alternatives. In fact, the public key size and the underlying one-time signature size are as large as in the single-signer setting. The authentication path increases by $\log N$ nodes as the associated Merkle tree consists of $N \cdot B$ leaf nodes, resulting in the same number of signatures per group member. This coincides with the number of signatures in the single-signer setting.

To realize this functionality we exploit the structure of Merkle trees [Mer90]. More precisely, we let all group members share a very same Merkle tree, which has the leaf nodes shuffled (by means of a block cipher or, more generally, a pseudorandom permutation (PRP)) before the tree is built. Our construction assigns a bounded number B of signatures to each group member. Each group member also has its own secret key. In Sect. 6, we give several options to handle a limitation related to the authentication paths, providing different trade-offs. We stress that none of these strategies seems to be optimal for all situations, but we hope that this discussion will feed further works in the community on how to optimally address this particular problem.

In terms of efficiency, for N group members, hash function digest size n , the size of our group signature is $|\text{one-time signature}| + n \cdot (\log N + \log B)$ bits, which is as large as a Merkle tree signature with $N \cdot B$ leaf nodes. Lattice-based counterparts occupy at least $\log N \cdot \tilde{O}(n)$ bits (ring variant) or $\log N \cdot \tilde{O}(n^2)$

bits (matrix variant) and the group public key size increases by a factor of $\log N$. Additionally, the underlying lattice problem weakens by a factor of $\log N$ (e.g. $\text{SIVP}_{\log N \cdot \tilde{O}(n^2)}$, see [LNW15]). The nature of our construction immediately carries over to the running times for signing and verification. Note that other group signature constructions often rely on costly zero-knowledge proofs (using Fiat-Shamir) to establish the different features of the group signature scheme, whilst our construction inherits them from the Merkle tree structure for free. Our scheme can be instantiated using any (stateful) Merkle-like signature scheme, e.g. the XMSS Merkle tree scheme [BDH11, HBG18].

In terms of security, we give a proof for full-traceability within the well-known framework of Bellare et al. (BMW model) [BMW03] and anonymity according to Camenisch and Groth [CG05]. The latter is required for a private key based revocation mechanism, which is a desirable feature and enables the verifier to identify signatures issued by a revoked private key. We also discuss a Proof-of-Concept implementation to show its efficiency and scalability for a publicly available tree.

1.3 Group Signature Scheme by Chaum and van Heyst

Our construction can be seen as an improvement of the very first discrete-log based group signature scheme due to Chaum and van Heyst [Cv91]. In their first construction each group member is randomly assigned a number of public and secret keys of a secure signature scheme, where each group member stores its assigned set of secret keys. To open signatures, the group manager stores the group member's name for every issued key. The group public key is represented as the (random) concatenation of all public keys. It can be seen that the approach taken in [Cv91] can be extended to any regular signature scheme. However, in our construction we only have one single hash value as the group public key regardless of the number of signers and signatures. Furthermore, each signer just stores one single secret seed, out of which all one-time key pairs and leaf nodes are derived. To ensure anonymity and traceability at reduced costs, the group manager applies a pseudorandom permutation to shuffle the positions of the leaf nodes and to open signatures without storing a large list of names and respective keys. All these modifications can directly be applied to [Cv91].

1.4 Open Problems

Full-anonymity in the BMW model would lead to a public-key encryption scheme [CG05, AW04] solely based on the existence of one-way functions, i.e. the group signature scheme would also serve as a basis to build other public key cryptographic primitives. However, there is little hope to achieve such a result as the seminal work [IR89] by Impagliazzo und Rudich already showed that one cannot base secure key agreement protocols on one-way functions. Thus, a challenging open problem consists of modifying our scheme in order to satisfy stronger notions of full-anonymity. Obtaining and storing the authentication path, a requirement of Merkle tree constructions, seems a limitation of our work hard to

circumvent. Finally, we note that multi-tree approaches (e.g. [HRB13]) are not applicable, thus limiting the maximum attainable tree height.

1.5 Organization

Section 2 presents the preliminary concepts, Sect. 3 the background on group signature schemes and related security notions and Sect. 4 presents our construction. Section 5 details its security assessment, Sect. 6 discusses the authentication path computation, Sect. 7 its implementation aspects and Sect. 8 our conclusions.

2 Preliminaries

It is well known [Mer90, BDS08, BDH11] that it is possible to build secure digital signature schemes using only a *secure hash function*. This is an advantage in comparison to any other signing scheme, which require not only a secure hash function but also a hard underlying computational problem. In this sense, hash-based signature schemes achieve minimal security requirements. The concept of secure hash function is vague and requires some refinement. Below we recap three computational problems related to hash functions useful to assess the security of hash-based signature schemes.

A cryptographic hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ is an efficiently computable function $\mathcal{H} = (H, \text{HKGen})$, where $\text{HKGen}(1^n)$ outputs a hash function H and H maps on input m and an element $m \in \{0, 1\}^*$ to $H(m) \in \{0, 1\}^n$. Depending on the given application scenario it may be required to have a certain set of properties [Rog04].

One-wayness (OW) A hash function H is said to be one-way, if it is infeasible for a PPT-adversary to find a preimage m of a random image.

Collision Resistance (CR) A hash function H is said to be collision resistant, if it is infeasible for a PPT-adversary to find two distinct messages $m \neq m'$ that map to the same hash value, i.e., $H(m) = H(m')$.

Second Preimage Resistance (SR) A hash function H is said to be second preimage resistant, if it is infeasible for a PPT-adversary and a given pair $(m, H(m))$ to find another message $m \neq m'$ that maps to the same hash value $H(m) = H(m')$. We note that CR implies SR.

To provide λ bits of classical security against collision and pre-image attacks, a hash function needs to have digest size of at least $n = 2\lambda$ bits and $n = \lambda$ bits, respectively. To maintain the same levels of security in a post-quantum world, the digest of the hash function would need to be extended by a factor 3/2 (for collision) and by a factor 2 (for pre-image). This is due to speedups induced by Grover's search algorithm [Gro96] on a quantum computer. Note that this

quantum speedup is marginal when compared to the one obtained by Shor's algorithm [Sho94] against RSA/ECC cryptosystems.

Most hash-based signature schemes are either one-time (OTS) or multi-time signature (MTS) schemes. OTS schemes (such as Winternitz [Mer90] and W-OTS+ [Hül13]) have an important limitation: a private key must not be used to sign more than one message (if so, it loses its security guarantees). Due to this limitation, Merkle proposed a way to transform an OTS scheme into a MTS scheme solely based on hash functions. This is known as the Merkle tree signature scheme [Mer90] and offers a way to tie many one-time public keys into a single multi-time public key. In this sense, any of the signatures generated by the one-time private keys can be validated with a single (multi-time) public key.

The Merkle scheme uses a binary tree of height h that is built from 2^h one-time key pairs. The leaf nodes are computed as the hash of the one-time public keys. The inner nodes are computed as the hash of concatenated children nodes. This rule is used to build all inner nodes up to the root, which is the multi-time public key.

3 Foundations of Group Signature Schemes

In this section we introduce the different definitions and security notions associated to group signature schemes following the work of Bellare et al. [BMW03] for full-traceability and anonymity according to Camenisch and Groth [CG05], which describe a comprehensive set of properties.

The appropriate security model for anonymity in our setting is due to [CG05], since the adversary is only given access to the secret keys of corrupt users in contrast to [BMW03], where the adversary has full access to the secret keys of all group members. The former reflects circumstances where the adversary is either static or he is adaptive and the parties cannot erase data (in these cases full-anonymity would not enhance security). In our scheme, we do not achieve full-anonymity in the sense of [BMW03] for an adaptive adversary and parties with erasing capabilities, since revealing the secret keys immediately allow to identify all the associated signatures. But this seems plausible for constructions solely based on the existence of one-way functions. We note that once our hash-based group signature scheme satisfies also full-anonymity following [BMW03] it is possible to build a public-key encryption scheme out of it. This would be a great result in post-quantum cryptography in general as this would imply public key cryptography (see Sect. 8) solely based on the security of one-way functions (minimal assumptions). In Sect. 8, we briefly discuss how full-anonymity allows to construct a public key encryption scheme following [CG05, AW04]. Thus, in the security game [CG05] the adversary is given a random signature and he must output the correct identity, under which it has been signed and which has not been corrupted before. We account for the fact that stateful Merkle-like schemes output a bounded number of signatures in the anonymity game.

In a group signature scheme there are essentially 3 parties involved:

- **Group Manager** He instantiates the scheme and generates the group public key. He assigns each group member with a secret key. In case of misuse of group signatures or misbehavior the group manager has the power to reveal the identity of a group signature by means of its master key.
- **Group Member** A group member can sign any data using its secret key such that his identity is concealed from any verifier other than the group manager.
- **Verifier** Any verifier can use the group public key in order to verify a group signature. He only knows that a group member signed the data, but he cannot specify which group member.

The syntax of a group signature scheme and the involved algorithms is as follows.

Syntax: A group signature scheme is composed by the following polynomial-time algorithms $\mathcal{GS} = (\text{G.KGen}, \text{G.Sign}, \text{G.Verify}, \text{G.Open})$.

G.KGen($1^k, 1^N$): The group key generation algorithm is a randomized algorithm that takes as input the security parameter k , the number of users N and generates and outputs a group public key gpk , the group signing keys gsk_i associated to the i -th group member for $i \in [N]$ and the group master key or tracing key gmsk required to open signatures by the group manager.

G.Sign(gsk_i, m): The group signing algorithm takes as input a group signing key gsk_i , a message $m \in \{0, 1\}^*$ and outputs a group signature σ on the message.

G.Verify(σ, m, gpk): The deterministic group verification algorithm takes as input a group signature, a message and the group public key, and outputs 1 in case the signature is valid, else 0.

G.Open(gmsk, σ, m): The group opening algorithm is a deterministic algorithm that on input the group master key, a signature, and the corresponding message outputs an identity related to σ .

There are two basic conditions to be satisfied in order for the scheme to work appropriately. In particular, the correctness requirement of the verification and tracing procedure has to be guaranteed for all honestly generated signatures. That is, for any group member $i \in [N]$ the following two expressions have to hold except with negligible probability

$$\text{G.Verify}(\text{G.Sign}(\text{gsk}_i, m), m, \text{gpk}) = 1$$

$$\text{G.Open}(\text{gmsk}, \text{G.Sign}(\text{gsk}_i, m), m) = i.$$

The first requirement mainly implies that all honestly generated group signatures must be valid. And the second expression allows the group manager by means of the master key to recover the identity of a correctly generated signature.

We now recap the security notions related to group signature schemes introduced in [BMW03] by Bellare et al. and subsequently relaxed in [BBS04] by Boneh et al. In the relaxed version, the adversary is not permitted to have oracle access to the opening procedure. For anonymity we refer to the security model of

Camenisch and Groth [CG05], where the adversary is indeed given access to the secret keys of corrupted group members. This model particularly also captures the possibility to realize private key based revocation, which represents a useful feature in remote attestation protocols as it may be a required feature to identify all signatures of an identity once its secret key gets exposed (e.g. extracted from the TPM) such that a potential adversary is prevented from signing under this identity. Following these models [BMW03, CG05], a group signature scheme is required to ensure two main security features, which we expound below.

3.1 Anonymity

The adversary not in possession of the group master key is not able to unveil the identity of a group member from its group signature. In the respective security game following [CG05] the adversary is given opening access in order to allow the adversary to see the identity of opened signatures. However, these models need to be modified in order take into account the constrained number B of signatures that a group member is able to issue. Therefore, the adversary is allowed to make arbitrary many calls to the opening and signing oracle for corrupted parties (at most B calls). For honest parties the adversary is only allowed to open at most $B - 1$ signatures per signer.

We differentiate between SPRP-Anonymity, which strictly follows the anonymity game of [CG05], and PRP-Anonymity, where the adversary is not allowed to have access to the opening oracle. Later, we will show that a pseudorandom permutation such as a secure block cipher can be used to instantiate the scheme satisfying either anonymity notions (Fig. 1).

Experiment $\text{Exp}_{\mathcal{G}\mathcal{S}, \mathcal{A}}^{\text{SPRP}, \text{an}-b}(k, N, B)$

$(\text{gpk}, \text{gsk}, \text{gmsk}) \leftarrow \text{G.KGen}(1^k, 1^N)$

$(\text{st}, i_0, i_1, m) \leftarrow \mathcal{A}^{\text{G.Open}(\text{gmsk}, \cdot, \cdot), \text{G.Sign}(\text{gsk}, \cdot, \cdot), \text{Corrupt}(\cdot)}(\text{choose}, \text{gpk})$

$\sigma \leftarrow \text{G.Sign}(\text{gsk}_{i_b}, m)$

$j \leftarrow \mathcal{A}^{\text{G.Open}(\text{gmsk}, \cdot, \cdot), \text{G.Sign}(\text{gsk}, \cdot, \cdot)}(\text{guess}, \text{st}, \sigma)$

- if \mathcal{A} queried the opening oracle on m, σ in the phase **guess**, return 0
- if \mathcal{A} queried i_0 or i_1 to **Corrupt**(\cdot), return 0
- if the maximal number of queries to **G.Sign**(gsk, \cdot, \cdot) wrt i_0 and i_1 is bounded by $B - 1$, return j else \perp .

Fig. 1. Experiment for SPRP-anonymity

Definition 1 (SPRP-Anonymity [CG05]). *Formally, a group signature scheme defined by the algorithms $\mathcal{G}\mathcal{S} = (\text{G.KGen}, \text{G.Sign}, \text{G.Verify}, \text{G.Open})$ is called anonymous, if for all probabilistic polynomial adversaries \mathcal{A} with access to the opening and signing oracles and all polynomially bounded N the advantage of the adversary in the experiment $\text{Exp}_{\mathcal{G}\mathcal{S}, \mathcal{A}}^{\text{SPRP}, \text{an}}(k, N, B)$ is negligible*

$$Adv_{\mathcal{G},\mathcal{A}}^{SPRP,an-b}(k, N, B) = |P[Exp_{\mathcal{G},\mathcal{A}}^{SPRP,an-1}(k, N, B) = 1] - P[Exp_{\mathcal{G},\mathcal{A}}^{SPRP,an-0}(k, N, B) = 1]|.$$

We also define a weaker form of anonymity taking into account the relaxation considered in [BBS04]. In particular, the adversary is not granted access to the opening oracle in the experiment. As we will see later, this will allow us to instantiate the scheme with only a secure block cipher.

Definition 2 (PRP-Anonymity). A group signature scheme defined by the algorithms $\mathcal{GS} = (G.KGen, G.Sign, G.Verify, G.Open)$ is called anonymous, if for all probabilistic polynomial adversaries \mathcal{A} with access to signing oracle and all polynomially bounded N the advantage of the adversary in $Exp_{\mathcal{G},\mathcal{A}}^{PRP,an}(k, N, B)$ is negligible

$$Adv_{\mathcal{G},\mathcal{A}}^{PRP,an-b}(k, N, B) = |P[Exp_{\mathcal{G},\mathcal{A}}^{PRP,an-1}(k, N, B) = 1] - P[Exp_{\mathcal{G},\mathcal{A}}^{PRP,an-0}(k, N, B) = 1]|.$$

See Fig. 2.

Experiment $Exp_{\mathcal{G},\mathcal{A}}^{PRP,an-b}(k, N, B)$

$(gpk, gsk, gmsk) \leftarrow G.KGen(1^k, 1^N)$

$(st, i_0, i_1, m) \leftarrow \mathcal{A}^{G.Sign(gsk, \cdot), Corrupt(\cdot)}(\text{choose}, gpk)$

$\sigma \leftarrow G.Sign(gsk_{i_b}, m)$

$j \leftarrow \mathcal{A}^{G.Sign(gsk, \cdot)}(\text{guess}, st, \sigma)$

if \mathcal{A} queried i_0 or i_1 to $Corrupt(\cdot)$, return 0

if the maximal number of queries to $G.Sign(gsk, \cdot)$ wrt i_0 and i_1 is bounded by $B - 1$, return j else \perp .

Fig. 2. Experiment for PRP-anonymity

3.2 Full-Traceability

This feature allows the group manager or possessor of the master key to revoke the anonymity of a signer and unveil its identity. Such a mechanism is important if misbehavior or misuse of the private key has been detected. In fact, this notion is even stronger as it is required that any set of colluding parties should not be able to create signatures that cannot be opened by the group manager or traced back to a group member, even if the parties have access to the master key, for instance extracted during key generation. We note that in this model we do not require to put a bound on the number of exchanged signatures.

Definition 3 (Full-Traceability [BMW03]). Formally, the group signature scheme $\mathcal{GS} = (G.KGen, G.Sign, G.Verify, G.Open)$ is called fully traceable, if for all probabilistic polynomial adversaries \mathcal{A} with access to the opening oracle and all polynomially bounded N the advantage of the adversary in the experiment $Exp_{\mathcal{G},\mathcal{A}}^{f-trace}(k, N)$ is negligible

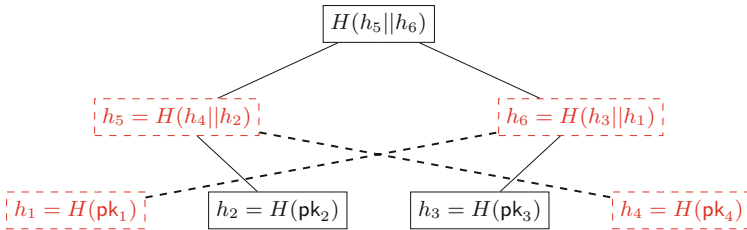
$$Adv_{\mathcal{G},\mathcal{A}}^{f-trace}(k, N) = P[Exp_{\mathcal{G},\mathcal{A}}^{f-trace}(k, N) = 1].$$

Experiment $\text{Exp}_{\mathcal{GS}, \mathcal{A}}^{f\text{-trace}}(k, N)$
 $(\text{gpk}, \text{gsk}, \text{gmsk}) \leftarrow \text{G.KGen}(1^k, 1^N)$
 $\text{st} \leftarrow (\text{gpk}, \text{gmsk})$
 $(m, \sigma) \leftarrow \mathcal{A}^{\text{G.Sign}(\text{gsk}, \cdot), \text{Corrupt}(\cdot)}(\text{guess}, \text{st})$
 if $\text{G.Verify}(\sigma, m, \text{gpk}) = 0$, return 0
 if i was queried to $\text{Corrupt}(\cdot)$, return 0
 if (i, m) was queried to $\text{G.Sign}(\text{gsk}, \cdot)$, return 0
 if $\text{G.Open}(\text{gmsk}, \sigma, m) = \perp \vee \exists i \in [N]$ with $\text{G.Open}(\text{gmsk}, \sigma, m) = i$
 return 1

Fig. 3. Experiment for full-traceability

4 G-Merkle: A Hash-Based Group Signature Scheme

Our stateful group signature scheme is based on the usage of a Merkle tree, as used in single-signer hash-based signature schemes. The core idea is to extend this approach to a multi-user setting, where more than one signer share the same tree in order to sign messages. A first attempt towards this direction consists in letting each user generate its own Merkle tree (as in the single-signer scheme). Then, each of those sub-trees could be appended to a super tree that will have as leaf nodes the root nodes of the sub-trees.



Hash-based Merkle tree in a multi signer setting, where the nodes are mixed. The nodes $\{pk_1, pk_2\}$ belong to Signer 1 and the nodes $\{pk_3, pk_4\}$ belong to Signer 2.

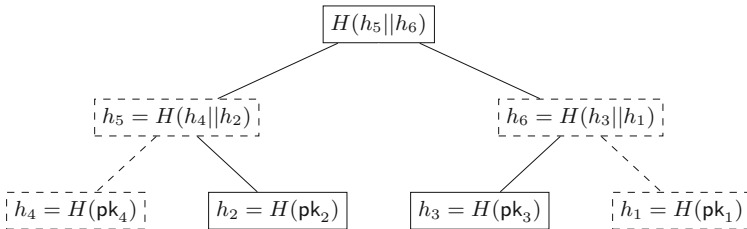


Fig. 4. Hash-based Merkle tree in a multi signer setting after shuffling the nodes. The nodes $\{pk_1, pk_2\}$ belong to Signer 1 and the nodes $\{pk_3, pk_4\}$ belong to Signer 2.

This naïve construction however does not meet the unlinkability property, a requirement of group signature schemes, since the authentication paths of all signatures issued by a certain signer would share at least one node (the root of its sub-tree). In order to overcome this obstacle, we apply a mixing strategy to the leaf nodes prior to the tree construction. That is, in the key generation phase, the group manager takes the set of leaf nodes from all parties and subsequently applies a secure uniform random permutation to the sets. That is, the permutation will mix the indices and hence the positions associated to the leaf nodes in the combined set. Subsequently, the super-tree is built as described before. In fact, this is a generic way of instantiating hash-based group signature schemes, which can be considered as an extension of the single-signer setting. Since our mixing strategy is based on the usage of pseudorandom permutations, we start with some required definitions. Let \mathcal{P}_n denote the set of permutations and $P \in \mathcal{P}_n$ with $P : \{0, 1\}^n \rightarrow \{0, 1\}^n$.

Definition 4. A pair of functions (E, D) with $E, D : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ is called (t, ϵ) pseudo random permutation, if E_r and D_r are inverses of each other for every $r \in \{0, 1\}^k$ and for any probabilistic polynomial adversary the success probability to distinguish a pseudorandom permutation from a truly random permutation is given by

$$Adv_{\mathcal{A}}^{Dist}(k, N) = |P_K[\mathcal{A}^{E_K, D_K}(\cdot) = 1] - P_{P \in \mathcal{P}_n}[\mathcal{A}^{P, P^{-1}}(\cdot) = 1]| \leq \epsilon$$

4.1 Instantiating PRPs from One-Way Functions

We note that while it is possible to instantiate pseudorandom permutations by use of block ciphers, pseudorandom permutations can particularly also be built from pseudorandom functions. More precisely, Luby and Rackoff [LR86] propose PRPs using pseudorandom functions combined with the Feistel construction. Goldreich et al. [GGM86] showed that pseudorandom generators imply pseudorandom functions, which in turn can be derived from any one-way function [HILL93]. This shows, that one-way functions indeed suffice to construct secure PRPs. Below we give a very simple way to generate pseudorandom permutations from pseudorandom functions.

Theorem 1 (Theorem 3.1, [NR96]). Let f_1, f_2 be independent pseudo random functions of length n and p_1, p_2 independent permutations of length $2n$. Define the functions

$$\begin{aligned} PRP(p_1, f_1, f_1) &= T_{f_1} \circ T_{f_2} \circ p_1 \\ SPRP(p_1, p_2, f_1, f_1) &= p_2^{-1} \circ T_{f_1} \circ T_{f_2} \circ p_1, \end{aligned}$$

where $T_{f_i}(l, r) = (r, l \oplus f_i(r))$ for $|l| = |r| = n$ and f_1, f_2, p_1, p_2 are chosen independently. Then, $PRP(p_1, f_1, f_1)$ is a pseudorandom function and $SPRP(p_1, p_2, f_1, f_1)$ is a strong pseudo random permutation.

By use of a suitable pseudorandom function, we can instantiate the group signature scheme with a secure SPRP with the aid of Theorem 1. In this case, the whole group signature scheme is just based on the existence of one-way functions, the minimal requirement for the existence of public key cryptography.

4.2 Instantiating PRPs from Block Ciphers

In general, one could apply a perfect uniform random permutation, where a permutation is chosen uniformly at random from a set of $n!$ elements. However, selecting an element from such a huge set requires at least $O(n \log(n))$ bits. For $n > 2^{20}$, this approach is impractical.

Thus, in practice, it is more efficient to instantiate pseudorandom permutations by means of block ciphers such as AES, SIMON and many others that satisfy the conditions from Definition 4. More specifically, the functions $E_K(\cdot)$ and $D_K(\cdot)$ correspond to the encryption and decryption functions of the respective block ciphers. Block ciphers represent a subset of all possible permutations.

Security of Block Cipher. When instantiating the scheme with a secure block cipher it is essential for anonymity that an adversary seeing a number T of (leaf position, group signer)-pairs ($T > 0$ for SPRP-anonymity and $T = 0$ for PRP-anonymity) cannot correctly map leaf positions to group members for the remaining leaves with non-negligible advantage. For instance, if a permutation is sampled from the set of all possible permutations, each group member may be associated to a remaining leaf supposing he did not issue all its signatures. In practice, this means that either a permutation is chosen uniformly at random from the set of all permutations (e.g. for a tree with 4 nodes) or the bit security of the block cipher is larger than or equal to the target security level of the scheme.

Block Ciphers with Larger Output Sizes. In practice, one does not find block ciphers permuting 10-bit or 20-bit integers (as it would be needed for $h = 10$ or $h = 20$) with security more than 100 bits. In this case, one can use larger block ciphers such as AES-128 or AES-256. The tree is then built slightly different. Once the manager receives all leaf nodes, it generates the set of tuples $S = \{(\text{leaf}_1, E_K(1)), \dots, (\text{leaf}_{2^h}, E_K(2^h))\}$ which contains leaves and the associated encrypted positions (128-bit or 256-bit integers, which are larger than the number of leaves 2^h). For instance, $(\text{leaf}_1, \dots, \text{leaf}_B)$ belong to group member 1 and $(\text{leaf}_{B+1}, \dots, \text{leaf}_{2B})$ to group member 2 and so on. Subsequently, the manager sorts the elements of S in an increasing order with respect to the second component. The new order represents the new positions of the leaves in the shuffled tree. The first layer of nodes is then built by not only including the leaves in the hashes but also the encrypted values of the respective leaves, e.g. $h_{i,j} = H(\text{leaf}_i, E_K(i) || \text{leaf}_j, E_K(j))$. All other tree layers up to the root are built as usual without any further modification. Due to this change, the encrypted

indices are part of the authentication path and the group manager can thus open signatures in case of misbehavior. For security, an adversary only sees 2^h encrypted indices somehow mapped to the initial positions (does not even know the index-ciphertext pairs). In the worst-case, the security of the block cipher will not decrease by more than $\log 2^h$ bits.

4.3 Our Construction: (Stateful) G-Merkle

In this section, we employ our group signature scheme on any Merkle tree based signature scheme. However, we keep our construction as general as possible by not restricting to any specific one-time signature scheme. In what follows, let $\mathcal{S} = (\text{KeyGen}, \text{Sign}, \text{Verify})$ denote the set of algorithms applied in a regular Merkle tree signature scheme.

G.KGen($\mathbf{1}^k, \mathbf{1}^N$): The group manager generates the master key $\text{gmsk} \in \{0, 1\}^k$ and initializes a block cipher ($E_{\text{gmsk}}(\cdot), D_{\text{gmsk}}(\cdot)$). Each user $i \in [N]$ is assigned a random secret key gsk_i associated to a secure one-time signature scheme such as Winternitz for B (e.g. $B = 2^t$) leaf nodes. That is, the group manager invokes N times G.KGen as in a regular Merkle tree signature scheme, however outputting only the secret key gsk_i and the hashed public keys serving as leaf nodes. The group manager proceeds as follows:

1. The set of indices associated to the leaf nodes of all users is shuffled

$$\text{Shuffle}(1, \dots, N \cdot B) = (j_1, \dots, j_{N \cdot B})$$

where $j_s = E_{\text{gmsk}}(s)$ for $s \in [N \cdot B]$. For instance, leaf node 1 is placed into position j_1 in the tree (see Fig. 4).

- 2 Subsequently, the group manager builds the G-Merkle tree on top of the shuffled set of nodes and generates the group public key gpk , which is represented as the root node of the G-Merkle tree.
3. Finally, the group manager transfers to user i the set of permuted indices

$$S_i = \{j_{(i-1)B+1}, \dots, j_{i \cdot B}\}$$

associated to the user's leaf nodes. As for the authentication path, the group manager can choose from several options to compute the authentication path for a signature. We refer to Sect. 6 for an overview.

G.Sign(gsk_i, m): User i maintains a counter t and a list of tuples

$$\text{state} = \{((i-1)B+1, E_{\text{gmsk}}((i-1)B+1), \dots, (i \cdot B, E_{\text{gmsk}}(i \cdot B)))\}$$

defining the possible states in the signing process. Whenever the user wishes to sign a message, he takes the actual state

$$\text{state}[t] = [(i-1)B+t, E_{\text{gmsk}}((i-1)B+t)]$$

from the list and sets $t := t + 1$, where the first component serves to internally identify the node with its associated secret key and the second component defines the position of the leaf node within the G-Merkle tree in order to deduce the appropriate authentication path. Supposing that the j -th leaf has been used to sign, following the authentication path consists of the nodes (a_0, \dots, a_{h-1}) , where h is the tree height. Define $\ell := \lfloor j/2^h \rfloor$ and denote by $v_j[k]$ the k -th node at the j -th layer, then we have

$$a_j = \begin{cases} v_j[\ell - 1] & \text{for } \ell \equiv 1 \pmod{2} \\ v_j[\ell + 1] & \text{for } \ell \equiv 0 \pmod{2}. \end{cases}$$

Finally, by use of the secret key \mathbf{gsk}_i the signing algorithm outputs a group signature (σ, m) on a message m that is composed by a one-time signature produced by the underlying signing algorithm **Sign** and the authentication path (see Sect. 6).

G.Verify $(\sigma, m, \mathbf{gpk})$: The deterministic group verification algorithm invokes **Verify** of the underlying signature scheme on the G-Merkle tree taking as input a group signature σ , a message m and the root node of the G-Merkle tree.

G.Open $(\mathbf{gmsk}, \sigma, m)$: On input the signature containing the exact authentication path, which can be represented by the position of the leaf node and the intermediate nodes, the manager extracts the position l and invokes the decryption algorithm $D_{\mathbf{gmsk}}(l) = j$ for $l \in [N \cdot B]$, otherwise he outputs \perp and aborts. Subsequently, he identifies the set S_i with $|S_i| = B$ s.th. $j \in S_i$ and outputs i .

Possible Modifications. We note that in practice one rarely finds PRPs with small output sizes. In this case, one applies our simple modifications from Sect. 4.2 using any secure blockcipher. We further note that one may adopt a dynamic approach in the key generation phase, where the group members individually generate their secret keys and associated leaf nodes. Subsequently, the leaf nodes are handed over to the group manager who shuffles the leaves and builds the Merkle tree on top of this mixed set. Such a strategy prevents the group manager from knowing the secrets and the risk of attacking the secrets is minimized.

5 Security

In this section, we prove that the construction proposed in Sect. 4 provides anonymity following [CG05] and full-traceability according to [BMW03]. Unforgeability of the scheme follows directly from the underlying signature scheme.

5.1 Anonymity

For anonymity, we prove the variant of Definition 1 where the adversary is allowed to have access to the secret keys of corrupt group members. The adversary must guess under which honest identity the signature has been created. Depending on the circumstance that the adversary is given or refused access to the opening oracle in the `choose` stage, we require the underlying block cipher to be an SPRP or PRP.

Theorem 2. *Let $N \cdot B$ be the number of leaves in the G-Merkle tree and N the number of users. Then, the construction described in Sect. 4.3 provides SPRP-anonymity following the experiment in Definition 1 under the assumption that a strong pseudorandom permutation is employed to shuffle the leaf nodes.*

Proof. The proof of this theorem is very simple as the adversary represents now an SPRP adversary involved in an indistinguishability game, however not having access to the encryption oracle. That is, assume there exists an adversary that breaks the anonymity of the scheme, then we can build an algorithm that distinguishes the encryption of different plaintexts derived under a strong pseudorandom permutation. In particular, the adversary is initially given access to the signing oracle and the opening oracle, which essentially represents the decryption algorithm, on group signatures of its choice during the `choose` stage. Due to the constrained tree size, the adversary can query the signing (and hence opening oracle on different signatures) at most $B - 1$ times for each honest member such that each user still has the chance to generate one last signature. He can also corrupt arbitrary group members. Eventually, the adversary outputs some state information st , an arbitrary message m to be signed and two existing identities i_0, i_1 , that have not been corrupted. In the second stage, the adversary gets as input the state information st and a signature σ on m under an identity, which is selected uniformly at random from i_0 and i_1 . The adversary is now challenged to make a right guess on the identity used to sign the message m . In this stage, the adversary still has access to the signing and opening oracle on signatures other than σ . The condition that each user can still sign a message prevents the adversary to exhaust all leaf nodes and make a trivial guess via exclusion based on the remaining signature not queried to the opening oracle yet.

The only element of a group signature that depends on the identities is the position of a leaf in the G-Merkle tree. All other elements can be replaced by the corresponding distributions that are independent from the identities. According to our construction from Sect. 4.3, the index set of the leaf nodes owned by signer S_k is $\{k_0, \dots, k_{B-1}\} = E_{\text{gmsk}}(\{(k-1) \cdot B, \dots, k \cdot B - 1\})$. We can safely assume that all but 1 plaintext-ciphertext pair per honest identity have been revealed (each signer can still sign one last message) such that the remaining plaintexts $p_0 \in [N \cdot B]$ and $p_1 \in [N \cdot B]$ of i_0 and i_1 are known to the adversary. The output of the challenger is a random ciphertext c that either encrypts p_0 or p_1 . Under the SPRP assumption of the cipher, we can replace the ciphertext set, in particular c as well, by random elements independent from the plaintext or

indices related to an identity. As a result, the claim follows. We note that if a perfect permutation was used to encrypt the indices, then the probability to have a particular plaintext is equal for any ciphertext given by the challenger. \square

In case the adversary is not granted access to the opening oracle in the experiment from Definition 2, we can even prove PRP-anonymity with only the requirement of using a pseudorandom permutation (analogously to Theorem 2).

Theorem 3. *Let $N \cdot B$ be the number of leaves in the G-Merkle tree and N the number of users. Then, the construction described in Sect. 4.3 provides PRP-anonymity in accordance to the experiment in Definition 2 assuming a pseudorandom permutation is employed to shuffle the leaf nodes.*

5.2 Traceability

Full-traceability subsumes a collection of other properties as described in [BMW03]. Following the traceability experiment given in Fig. 3, the proof relies on the unforgeability of the underlying signature scheme. In fact, our G-Merkle scheme inherits its existential unforgeability immediately from the basic scheme as described in Sect. 2. In general, if the basic Merkle tree construction is secure against such an adversary, then so is the G-Merkle tree construction. Thus, we call our group signature scheme G-Merkle_{XMSS} if it relies on XMSS.

Theorem 4. *Let T be a hash-based one-time/few-time signature scheme and Merkle_T the corresponding Merkle tree based multi-time signature scheme. If Merkle_T is existentially unforgeable under chosen message attacks, then so is the group signature scheme G-Merkle_T.*

The proof of this statement is straightforward as the multisigner G-Merkle_T scheme differs from a single-signer Merkle_T scheme in the shuffling procedure, where the order of the nodes is changed, and the number of participants with their own secret keys.

Theorem 5. *Let T be the number of leaves in G-Merkle as defined in Sect. 4. Suppose that there exists a PPT traceability adversary, then there exists an algorithm \mathcal{B} that breaks the unforgeability of the underlying signature scheme.*

Proof. The proof of this theorem is mainly based on Theorem 4 stating the existential unforgeability of the underlying digital signature scheme. We show that such an adversary does not exist unless the underlying signature scheme is insecure. We proceed by means of the experiment defined in Fig. 3 following [BMW03].

Suppose, there exists an attacker that can successfully generate such a forgery. Clearly, in the G-Merkle tree construction the height of the tree $\log_2(N \cdot B)$ and number of leaves $N \cdot B$ is known in advance such that a group signature can only be valid if the index of the leaf node used to sign the message is an element of $\{0, \dots, N \cdot B\}$. Therefore, the adversary must produce a

forgery that opens to an identity of an honest user $i \notin \mathcal{C}$, which correctly verifies. However, this is only the case if the attacker breaks the unforgeability of the underlying signature scheme (Merkle tree construction), since he is not in possession of the secret key associated to the identity i . Due to Theorem 4 such an adversary does not exist. \square

6 Authentication Path Computation

The G-Merkle tree is composed by leaf nodes originating from different users. Thus, the conventional approach of generating the authentication path is not immediately applicable as the authentication path inherently requires the knowledge of the other nodes. As a result, we need a different strategy in order to derive the authentication path. In what follows, we propose some possible solutions to tackle this target.

Public Leaf Nodes. The first approach works similar to the very first Merkle tree constructions, where a user stores each leaf node of the associated tree. Translating this strategy to the multi-user setting the group manager publishes all leaf nodes of all users. This leads to a storage size of at most $N \cdot B$ nodes. Whenever a user invokes its signing algorithm it combines the leaf nodes in order to generate the authentication path associated to its one-time signature. Alternatively, the whole tree can be published/stored and the memory requirement just double. In this case the running times for signing decrease as group members are no longer required to compute the inner nodes.

User Directed Authentication Path Computation. The group manager can also send to each user the authentication paths together with the associated indices during key generation. The user stores the nodes and can delete those nodes once they are consumed.

Lemma 1. *Let N denote the number of group members and B the number of potential signatures per user. Then the memory size Mem of a user is bounded by*

$$\log N \leq \text{Mem} \leq B \cdot \log(N) + B < N \cdot B$$

Proof. The best possible case occurs when all nodes of a user are neighbors such that he can generate many of the entries in the authentication path by use of his key pair. However, in this case, he can build a subtree of height $1 + \log_2 B$ and requires to store $\log N$ nodes in order to build the authentication path. In the worst-case the user stores the nodes of the $\log B$ -th layer, i.e. B nodes, which allow him to generate all other nodes in the upper part of the G-Merkle tree. This is due to the fact that all B potential signatures have to cross one of the B nodes in the $\log B$ -th layer. Furthermore, he has to store at most $B \cdot \log N$ nodes from the remaining $\log N$ layers of the tree, which corresponds to the nodes in the authentication path. \square

In practice each user can determine the exact number of nodes to be stored and hence optimize the memory size. He can eliminate duplicated nodes that appear in multiple authentication paths. For instance, all authentication paths have to use either the left or right child of the root node. The user is doing best if he chooses to store both right and left children (if req.) only once.

Improved Storage Size with Clustering. Based on the proof of Lemma 1 and the observation that the memory sizes improve, if the leaf nodes of a group member are close to each other, it is possible to split the group into several clusters. This is accomplished secretly by the group manager. The leaf nodes of the G-Merkle tree are then clustered accordingly. For instance, we partition the leaf nodes into k clusters, where each cluster contains all nodes of N/k users. This enhances the probability for each group member to use many of the nodes in the authentication path several times such that the absolute storage requirement is reduced. Due to the fact that the verifiers and group members themselves are not aware of how the leaf nodes are partitioned, how many clusters do exist and who are the group members within a particular cluster, the anonymity is still guaranteed in case the adversary is not given access to the opening oracle within the anonymity game. Such a clustering strategy is advantageous if N is very large. A further advantage of the clustering strategy is the usage of block ciphers of small output sizes. In fact, the output size can be chosen equal to the cluster size.

In case the adversary is given access to the opening oracle, the adversary at most learns which users belong to a cluster. This reduces anonymity in the whole set of group members to anonymity of the smaller cluster. But the adversary is not able to map a signature to a certain group member of a cluster in the challenge phase. Assuming that $N = 2^{t_1}$ and the cluster size is $k = 2^{t_2}$, the authentication path of users from a same cluster have always the same last $\log(N \cdot B) - \log(N \cdot B/k) = t_2$ nodes in the authentication path. This is due to the fact that a certain cluster has the same parent node at height $\log(N \cdot B/k)$. From then on, the sibling nodes are identical.

Interactive Authentication Path Computation. In case the group manager maintains a list of the group signers secret keys, it is possible to ask the group manager the required authentication path in an online fashion. Once a group member sends its part of the signature (OTS) together with the leaf position to the verifier, the verifier can invoke the group manager for the associated authentication path. Clearly, this has no impact on the security as all leaf nodes can be made public.

7 Implementation

In this section, we discuss the implementation aspects of our proposal. This discussion is based on our G-Merkle implementation in C, which is an extension

of a XMSS/WOTS+ implementation as specified in [HBGM18], using SHA2-256 as the hash function.

Our implementation follows the approach described in Sect. 4.2 where block ciphers with larger outputs are used. More precisely, we use AES-256 to perform the indices encryption (which acts as the shuffling process of the leaf nodes). Therefore, the leaf nodes indices are initially represented as 256-bits long integer numbers (padded with zeros on the left) and then encrypted using AES-256. The 256-bits ciphertexts are considered as the new (shuffled) leaf node indices. Note that most of the encrypted indices will likely be out of the range $[0, 2^h - 1]$, but (as described in Sect. 4.2) G-Merkle only cares about the ordering in which these encrypted indices appear. These encrypted indices are also used to open signatures. Once the encrypted indices are computed, they are sorted in increasing order. Our implementation uses a simple Quicksort implementation for very large (256-bits long) numbers. We note that speedups in this step might be achieved by using other sorting algorithms (e.g. Radix sort).

Table 1 shows the performance data of the G-Merkle inner processes. We fix $N = 64$ as the number of group members (also called users) to facilitate the comparison of different tree heights (but other values are possible, given the trade-off between group members and number of signatures). The performance data are given in thousands of cycles measured in an Intel(R) Core(TM) i5-63000 CPU @2.40 GHz with 16 GB of RAM. The code has been compiled with GCC 6.4.0 with -O3 compilation flag. Each process has been repeated 100 times and the number of cycles averaged. The first column represents the relevant processes in G-Merkle, namely the generation of the leaf nodes (each user has to perform this step, i.e. generate all OTS key pairs and the corresponding leaf nodes), the encryption and sorting of all indices, the Merkle tree building process, XMSS signature generation, XMSS signature verification and signature opening (which consists of a call to AES-256 decryption). Between parentheses, we denote whether the operation is performed by each user (U) or only by the group manager (GM). The most expensive operation consists of building the tree, an operation handled by the group manager only, not impacting the users.

Table 1. G-Merkle performance (in kcycles). U = User, GM = Group Manager.

Process (owner)	$N = 64$		
	$(h = 14, B = 256)$	$(h = 16, B = 1024)$	$(h = 18, B = 4096)$
Generate leaf nodes (U)	2, 319, 508	9, 302, 171	35, 646, 646
Encrypt indices (GM)	56, 960	225, 818	934, 001
Sorting (GM)	16, 866	85, 767	364, 334
XMSS tree building (GM)	24, 347, 871	114, 011, 307	440, 567, 352
XMSS sign (U)	7, 052	7, 153	7, 059
XMSS verify (U)	9, 007	9, 092	9, 398
Signature opening (GM)	100	99	102

The actual group management operations, such as encryption and sorting of the leaf node indices, and opening of signatures, do not represent any significant overhead, while the XMSS algorithms (sign and verify) are efficient. Our implementation assumes that the Merkle tree is publicly (and securely) available, as discussed in Sect. 6, thus the authentication path computation (a somewhat expensive operation in XMSS) is not relevant here.

8 Conclusion and Discussion

We introduced the first (stateful) hash-based group signature scheme and showed that it is based on standard assumptions and not on expensive non-interactive zero knowledge proof systems, as seen in other group signature schemes. Our approach exploits the structure of Merkle trees in general. Due to this fact, we can generate group signatures more efficiently in terms of running times, signature and key sizes. It is worth mentioning that the provisioning of the authentication paths is challenging. By merging different trees from different users and randomizing the tree structure, the simplicity to generate authentication paths as in XMSS is lost at the benefit of anonymity. We presented several ways to obtain the authentication path (such as publishing the whole Merkle tree), but stress that it is still an open problem how to address it optimally. Furthermore, we strongly emphasize that full-anonymity would result in a secure public key encryption scheme. To illustrate the transformation of a group signature scheme into an encryption scheme, once full-anonymity is achieved, the group manager generates the master key and all secret keys of the different “identities”. The group manager publishes the secret keys as the public key and keeps the master key secret. If a party wishes to encrypt a message, he encodes this message in terms of an identity (or identities) and sends the signature to the group manager, who in turn decrypts the ciphertext by opening the signature. He reveals the identity, which represents the encoded message. Due to full-anonymity an adversary cannot unveil the identity given all secret keys of the identities. Such a result would have a huge impact in cryptography in general as it would allow to build public key cryptography solely based on the existence of one-way functions. This however would somehow oppose the results of [IR89], which states that one-way functions are not sufficient to build key agreement protocols.

Acknowledgements. We thank Andreas Hülsing for engaging in helpful discussions and the anonymous reviewers for providing detailed comments.

References

- [ACJT00] Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 255–270. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_16
- [AW04] Abdalla, M., Warinschi, B.: On the minimal assumptions of group signature schemes. In: Lopez, J., Qing, S., Okamoto, E. (eds.) ICICS 2004. LNCS, vol. 3269, pp. 1–13. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30191-2_1
- [BBS04] 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
- [BDH11] Buchmann, J., Dahmen, E., Hülsing, A.: XMSS - a practical forward secure signature scheme based on minimal security assumptions. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 117–129. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_8
- [BDS08] Buchmann, J., Dahmen, E., Schneider, M.: Merkle tree traversal revisited. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 63–78. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88403-3_5
- [BL09] Brickell, E., Li, J.: Enhanced privacy ID from bilinear pairing (2009)
- [BMW03] Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_38
- [BSZ05] Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: the case of dynamic groups. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 136–153. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30574-3_11
- [BW06] Boyen, X., Waters, B.: Compact group signatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 427–444. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_26
- [BW07] Boyen, X., Waters, B.: Full-domain subgroup hiding and constant-size group signatures. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 1–15. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71677-8_1
- [CG05] Camenisch, J., Groth, J.: Group signatures: better efficiency and new theoretical aspects. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 120–133. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30598-9_9
- [CL02] Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_5
- [CL04] Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_4

- [Cv91] Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_22
- [DP06] Delerablée, C., Pointcheval, D.: Dynamic fully anonymous short group signatures. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 193–210. Springer, Heidelberg (2006). https://doi.org/10.1007/11958239_13
- [ELL+15] Ezerman, M.F., Lee, H.T., Ling, S., Nguyen, K., Wang, H.: A provably secure group signature scheme from code-based assumptions. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 260–285. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_12
- [GGM86] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. *J. ACM* **33**(4), 792–807 (1986)
- [GKV10] Gordon, S.D., Katz, J., Vaikuntanathan, V.: A group signature scheme from lattice assumptions. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 395–412. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_23
- [GMO16] Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: faster zero-knowledge for Boolean circuits. In: 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, pp. 1069–1083. USENIX (2016)
- [Gro96] Grover, L.K.: A fast quantum mechanical algorithm for database search. In: 28th Annual ACM Symposium on Theory of Computing, Philadelphia, Pennsylvania, USA, 22–24 May 1996, pp. 212–219. ACM Press (1996)
- [Gro07] Groth, J.: Fully anonymous group signatures without random oracles. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 164–180. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76900-2_10
- [HBGM18] Hülsing, A., Butin, D., Gazdag, S., Mohaisen, A.: XMSS: extended hash-based signatures. Technical report, Internet Draft draft-irtf-cfrg-xmss-hash-based-signatures-12 (2018). <https://datatracker.ietf.org/doc/draft-irtf-cfrg-xmss-hash-based-signatures/>
- [HILL93] Hstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: Construction of a pseudo-random generator from any one-way function. *SIAM J. Comput.* **28**, 12–24 (1993)
- [HRB13] Hülsing, A., Rausch, L., Buchmann, J.: Optimal parameters for XMSS^{MT}. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CDARES 2013. LNCS, vol. 8128, pp. 194–208. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40588-4_14
- [Hül13] Hülsing, A.: W-OTS+ – shorter signatures for hash-based signature schemes. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 2013. LNCS, vol. 7918, pp. 173–188. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38553-7_10
- [IR89] Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: Proceedings of the 21st Annual ACM Symposium on Theory of Computing, 14–17 May 1989, Seattle, Washington, USA, pp. 44–61 (1989)
- [LLLS13] Laguillaumie, F., Langlois, A., Libert, B., Stehlé, D.: Lattice-based group signatures with logarithmic signature size. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 41–61. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_3

- [LLNW14] Langlois, A., Ling, S., Nguyen, K., Wang, H.: Lattice-based group signature scheme with verifier-local revocation. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 345–361. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_20
- [LNW15] Ling, S., Nguyen, K., Wang, H.: Group signatures from lattices: simpler, tighter, shorter, ring-based. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 427–449. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_19
- [LR86] Luby, M., Rackoff, C.: How to construct pseudo-random permutations from pseudo-random functions. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, p. 447. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_34
- [Mer90] Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_21
- [MKF+16] McGrew, D., Kampanakis, P., Fluhrer, S., Gazdag, S.-L., Butin, D., Buchmann, J.: State management for hash-based signatures. In: Chen, L., McGrew, D., Mitchell, C. (eds.) SSR 2016. LNCS, vol. 10074, pp. 244–260. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49100-4_11
- [MV03] Micciancio, D., Vadhan, S.P.: Statistical zero-knowledge proofs with efficient provers: lattice problems and more. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 282–298. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_17
- [NR96] Naor, M., Reingold, O.: On the construction of pseudo-random permutations: Luby-rackoff revisited. IACR ePrint 1996:11 (1996)
- [NSA15] National Security Agency NSA: Commercial national security algorithm suite (2015). <https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm>
- [NZZ15] Nguyen, P.Q., Zhang, J., Zhang, Z.: Simpler efficient group signatures from lattices. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 401–426. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_18
- [oSN16] National Institute of Standards and Technology NIST: Post-quantum crypto project (2016). <http://csrc.nist.gov/groups/ST/post-quantum-crypto/>
- [PQC16] PQCrypto: Post-quantum cryptography for long-term security pqcrypto ict-645622 (2016). <https://pqcrypto.eu.org/>
- [Rog04] Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–358. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25937-4_22
- [Sho94] Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, 20–22 November 1994, pp. 124–134. IEEE Computer Society Press (1994)

Quantum Algorithms



Quantum Collision-Finding in Non-uniform Random Functions

Marko Balogh^{1(✉)}, Edward Eaton^{2(✉)}, and Fang Song^{3(✉)}

¹ Department of Physics, Portland State University, Portland, USA
marko.balogh@me.com

² Department of Combinatorics and Optimization,
University of Waterloo, Waterloo, Canada
eeaton@uwaterloo.ca

³ Department of Computer Science, Portland State University, Portland, USA
fsong@pdx.edu

Abstract. We study *quantum* attacks on finding a collision in a *non-uniform* random function whose outputs are drawn according to a distribution of min-entropy k . This can be viewed as showing *generic* security of hash functions under *relaxed* assumptions in contrast to the standard heuristic of assuming uniformly random outputs. It is useful in analyzing quantum security of the Fujisaki-Okamoto transformation [31]. In particular, our results close a gap left open in [30].

Specifically, let D be a distribution of min-entropy k on a set Y . Let $f : X \rightarrow Y$ be a function whose output $f(x)$ is drawn according to D for each $x \in X$ independently. We show that $\Omega(2^{k/3})$ quantum queries are necessary to find a collision in f , improving the previous bound $\Omega(2^{k/9})$ [30]. In fact we show a stronger lower bound $2^{k/2}$ in some special case. For most cases, we also describe explicit quantum algorithms matching the corresponding lower bounds.

1 Introduction

Hash functions are central and prominent in modern cryptography, and there have been many ingenious designs of cryptographic hash functions [2, 4, 13, 26]. One significant property of a cryptographic hash function H , backed with intensive tests in practice, is *collision resistance*. Namely, it should be computationally unfeasible to find a *collision*, which is a pair of distinct input strings (x, x') with $H(x) = H(x')$. Because of this nice feature, hash functions are being used in numerous cryptographic constructions and applications, e.g., protecting passwords [1], constructing message authentication codes and digital signature schemes, as well as various crypto-currencies exemplified by BitCoin [25].

Theoretical analysis of a hash function H often refers to *generic* security, where one ignores the internal design of H and views it as a black box. Moreover, the output of H is assumed to have been drawn *uniformly* at random from some codomain of size N . The complexity of finding a collision is then measured by

the number of evaluations of H , i.e., queries to the black box. By the well-known *birthday* bound, $\Theta(\sqrt{N})$ queries are both sufficient and necessary to find a collision in H . These principles are extended and formalized as the *random oracle* model, in which a hash function is treated as a truly random function that is publicly available but only through oracle queries [11]. This heuristic has been widely adopted to construct more efficient cryptosystems and facilitate security reduction proofs which are otherwise challenging or unknown [12, 21].

However, in reality, there are attacks that perform significantly better than the plain birthday attack. The recent explicit break of full SHA-1 by Google and the Cryptology Group at the Netherlands' Centrum Wiskunde & Informatica [29], in which two PDF files can be generated that collide on the same 160-bit digest, only takes $\sim 2^{61}$ hash evaluations instead of the 2^{80} expected via the birthday attack. These attacks are possible because the internal structure of H may create opportunities for more effective cryptanalysis. A natural reaction would be to figuratively open up the black box and take into account the inner workings case-by-case when analyzing a hash function. Alternatively, *can we prove generic security bounds, but under relaxed and/or more accurate assumptions?*

The approaching era of quantum computing will make these challenges more worrisome. The power of quantum computers, while promising in accelerating the resolution of fundamental problems in many areas such as chemistry, biology, etc., raises a tremendous threat to cryptography. Many public key cryptosystems will be broken due to Shor's efficient quantum algorithm for the factoring and discrete logarithm problems upon which they are based [27]. In addition, new features of quantum adversaries are difficult and subtle to deal with, especially in the setting of cryptographic protocols. In fact many classical security analyses become inapplicable or even fail completely in the presence of quantum adversaries [17, 23, 33].

Pertaining to hash functions, a quantum adversary is able to implement the hash function as a quantum circuit and evaluate it in quantum *superposition*. Therefore, if H is treated as a black box, it is reasonable to allow a quantum adversary to query H in quantum superposition: $\sum_x \alpha_x |x, 0\rangle \mapsto \sum_x \alpha_x |x, H(x)\rangle$. Although this does not imply that the adversary can learn the entirety of H in one query, an immediate difficulty, for example, is the failure of the "lazy sampling" trick, where one can simulate a random function by sampling random responses on-the-fly. Indeed, much effort has been devoted to extending the results and useful techniques in the classical random oracle model to the quantum setting (formalized as the quantum random oracle model) [9, 14, 19, 38]. Notably, Zhandry [37] shows that $\Theta(N^{1/3})$ quantum queries are both sufficient and necessary to find a collision in a uniformly random function. This establishes the generic security of uniformly random hash functions. But as classical attacks have illustrated, assuming uniform randomness is sometimes too optimistic and risky. Such concerns are becoming more pressing due to recent advances in the physical realization of quantum computers [3, 5]. Optimized architectures are also reducing the cost of implementing quantum algorithms (e.g., see an estimation of Grover's quantum search algorithm [10]).

This motivates the question we study in this work: *what is the complexity of finding a collision in a **non-uniform** random function, under quantum attacks in particular?* Specifically we consider a distribution D_k on set Y which has min-entropy k , i.e., the most likely element occurs with probability 2^{-k} . We want to find a collision in a function $H : X \rightarrow Y$ where for each $x \in X$, $H(x)$ is drawn independently according to D_k . We call it a rand-min- k function hereafter. Note that if D_k is uniform over Y (hence $|Y| = 2^k$), this becomes the standard uniformly random function. Given H as a black-box, we are interested in the number of queries needed by a quantum algorithm to find a collision in H . As a result, this will establish the generic security of hash functions under a *relaxed condition* where the outputs of a hash function are drawn from a distribution of min-entropy k rather than a strictly uniform distribution. This condition might be a more realistic heuristic for a good hash function. Roughly speaking, a hash function designer will only need to make sure that there is no single value $y \in Y$ that has a large set of preimages (i.e., $f^{-1}(y) := \{x \in X : f(x) = y\}$ with $|f^{-1}(y)| \leq 2^k$). In contrast, modeling a hash function as a uniformly random function would require certain *regularity* such that the preimage set of every codomain element has roughly the same size, which may be difficult to justify and test in practice. We also note that a concrete application of collision finding in rand-min- k functions appears in the famous Fujisaki-Okamoto transformation [21], whose quantum security has been studied in [31].

Classically, it is not difficult to derive a variation of the birthday bound, which gives $\Theta(2^{k/2})$ as the query complexity in typical cases. In the quantum setting, Targhi et al. [30] prove that $\Omega(2^{k/9})$ queries are necessary for any quantum algorithm to find a collision with constant probability. Compared to the tight bound $2^{k/3}$ in the uniform case, the bound is unlikely to be optimal and the gap seems significant. In addition, no quantum algorithms are described or analyzed formally. Overall, our understanding of finding a collision in non-uniform random functions is far from satisfying as far as quantum attacks are concerned.

1.1 Our Contributions

In this work, we characterize the complexity of finding collisions in a rand-min- k function when it is given as an oracle to a quantum algorithm. We are able to prove matching upper and lower bounds in many cases. The results are summarized in Table 1.

A simple special case is the flat distribution, which is uniform on a subset of size 2^k . In this case, not surprisingly, the same bound $2^{k/3}$ for the uniform random function holds. Another special case, which represents the hardest instances, concerns the δ -min- k distributions, where there is a mode element with probability mass 2^{-k} and the remaining probability mass is distributed uniformly throughout the rest of the codomain. Here we show that $2^{k/2}$ queries are both sufficient and necessary. For general min- k distributions, the complexity is characterized by the *collision variable* $\beta(D)$ for a distribution D , which is the reciprocal of the probability that two independent samples from D collide.

Table 1. Summary of quantum collision finding in rand-min- k functions. $\beta := \frac{1}{\Pr[x=y:x,y \leftarrow D]}$ is the collision variable, which equals 2^k for flat-distributions (i.e., uniform on a subset of size 2^k), and lies in $[2^k, 2^{2k}]$ for δ -min- k distributions (i.e., peak at one element, and uniform elsewhere), as well as for general min- k distributions. Here M refers to the size of the domain and N refers to the size of the codomain.

D_k	M, N, k settings	Upper bound	Lower bound	Match?
All	$M = o(\beta^{1/2})$ (inj. by Lemma 2)	∞	∞	✓
All	$M = \Omega(\beta^{1/2})$	$\beta^{1/3}$ (Theorem 5)	$2^{k/3}$ (Corollary 2)	✗
flat- k	$M = \Omega(2^{k/2})$	$2^{k/3}$ (Theorem 5)	$2^{k/3}$ (Corollary 2)	✓
δ -min- k	$M = \Omega(N^{1/2}), 2^k \leq N < 2^{3k/2}$	$N^{1/3}$ (Theorem 5)	$N^{1/3}$ (Corollary 3)	✓
	$M = \Omega(N^{1/2}), 2^{3k/2} \leq N < 2^{2k}$	$2^{k/2}$ (Theorem 6)	$2^{k/2}$ (Corollary 3)	✓
	$M = \Omega(2^k), N \geq 2^{2k}$	$2^{k/2}$ (Theorem 6)	$2^{k/2}$ (Corollary 3)	✓

We prove a generic upper bound $\beta^{1/3}$, and a lower bound $2^{k/3}$. For comparison, classically one can show that the (generalized) birthday bound $\Theta(\beta^{1/2})$, which equals $\Theta(N^{1/2})$ for uniform distributions, precisely depicts the hardness of finding a collision.

Technical overview. For the generic lower bound $2^{k/3}$, we follow the natural idea of reducing from collision finding in uniform random functions (Theorem 3). We show that finding a collision in a uniformly random function of codomain size 2^k reduces to that in flat distributions, and then to general min- k distributions. Therefore the $2^{k/3}$ lower bound follows. This approach is in contrast to that in [30], where they basically extract close-to-uniform bits from the output of a rand-min- k function f by composing f with a universal hash function h . Note that a collision in f is also a collision in $h \circ f$. In addition, $h \circ f$ can be shown to be quantum indistinguishable from a uniformly random function by a general theorem of Zhandry [36], which relates sample-distinguishability to oracle-distinguishability. Therefore any adversary for rand-min- k can be turned into an adversary for $h \circ f$, contradicting the hardness for uniformly random functions. However, the discrepancy between $h \circ f$ and a uniformly random function gets accumulated and amplified in the sample-to-oracle lifting step, and this may explain the slackness in their lower bound $2^{k/9}$.

Instead, given an oracle f whose images are distributed according to a distribution D , our reductions employ a *redistribution function* to simulate an oracle f' whose images are distributed according to another distribution D' on Y' . A redistribution function r maps a pair $(x, f(x))$ to an element in Y' , and r is sampled from a proper distribution such that $f'(x) := r(x, f(x))$ is distributed according to D' , taking into account the random choice of f as well. We show algorithms for sampling appropriate redistribution functions, called *redistribution function samplers*, for the distributions we are concerned with. As a result, we can use an adversary for the collision-finding problem in D' to attack the collision-finding problem in D . To complete the reductions, we show that a collision found in the simulated oracle for f' will indeed be a valid collision in f with probability at least $1/2$.

Along the same lines, it is possible to demonstrate that collision-finding in δ -min- k distributions is the hardest case. In fact, we are able to establish rigorously a *strengthened* lower bound in this case (Theorem 4). Our proof proceeds by showing indistinguishability between a random δ -min- k function on a codomain of size N and a uniformly random function on the same codomain. Then the lower bound in the uniform case translates to a lower bound for the δ -min- k case. The exact bounds vary a bit for different relative sizes of N and k .

Establishing upper bounds is relatively easy (Theorem 5). We adapt the quantum algorithm of [37] in the uniform case. Basically we partition the domain of a rand-min- k function f into subsets of proper size, so that when restricting f on each subset, there exists a collision with at least constant probability. Next, we can invoke the collision finding algorithm by Ambainis [8] on each restricted function, and with a few iterations, a collision will be found.

Moreover, we give alternative proofs showing the lower bound for δ -min- k distributions (Theorem 6). They are helpful to provide more insight and explain the bounds intuitively. Specifically, we reduce an average-case search problem, of which the hardness has been studied [24], to finding a collision in a δ -min- k random function. On the other hand, when the mode element of a min- k distribution is known, we show that applying Grover's quantum search algorithm almost directly will find a collision within $O(2^{k/2})$ queries. This actually improves the algorithms above in some parameter settings.

1.2 Discussion

Collision finding is an important problem in quantum computing, and a considerable amount of work in this context exists. Brassard et al. [16] give a quantum algorithm that finds a collision in any two-to-one function $f : [M] \rightarrow [N]$ with $O(N^{1/3})$ quantum queries. Ambainis [8] gives an algorithm based on quantum random walks that finds a collision using $O(M^{2/3})$ queries whenever there is at least one collision in the function. Aaronson and Shi [6] and Ambainis [7] give an $\Omega(N^{1/3})$ lower bound for a two-to-one function f with the same domain and co-domain of size N . Yuen [35] proves an $\Omega(N^{1/5}/\text{poly}(\log N))$ lower bound for finding a collision in a uniformly random function with a codomain at least as large as the domain. This is later improved by Zhandry [37] to $\Theta(N^{1/3})$ for general domain and codomain as we mentioned earlier.

We stress that, typically in quantum computing literature, the lower bounds are proven for the worst-case scenario and with constant success probability. This in particular does not rule out adversaries that succeed with an inverse polynomial probability which is usually considered a break of a scheme in cryptography. Hence a more appropriate goal in cryptography would be showing the number of queries needed for achieving any (possibly low) success probability, or equivalently bounding above the success probability of any adversary with certain number of queries. Our results, as in [30, 37], are proven in the strong sense that is more appropriate in cryptographic settings.

Our work leaves many interesting possible directions for future work. For some distributions, our reductions may take a long time to implement. Can we

find time-efficient reductions in general? We have been mainly concerned with finding one collision; it is interesting to investigate the complexity of finding *multiple* collisions in a non-uniform random function. Finally, we note that a stronger notion for hash functions called *collapsing* has been proposed which is very useful in the quantum setting [32]. Can we prove that rand-min- k functions are collapsing? Note that a uniform random function is known to be collapsing, and more recently it has been shown that the sponge construction in SHA-3 is collapsing (in the quantum random oracle model) [18].

Missing proofs and more. Due to space limitations, we omit a few proofs in this submission. The full version can be found at ia.cr/2017/688, where in addition to the missing proofs, we also extend the work here and give tight analysis for the quantum generic security of preimage and second-preimage resistance of hash functions under non-uniform output distributions.

Independent work. In a concurrent and independent work by Ebrahimi and Unruh [20], they give twelve bounds for quantum collision finding of min- k random functions. We observe that ten of them coincide with our bounds, and in particular, they present essentially the same quantum collision-finding algorithms as ours. The remaining two are generic lower bounds improving upon their prior work [30], which are $\Omega(2^{k/5})$ and $\Omega(\beta^{1/9})$ (in our notation). Our bounds are stronger – $\Omega(2^{k/3})$ and $\Omega(\beta^{1/6})$ (by noting that $\beta \leq 2^{2k}$) respectively.

2 Preliminaries

Here we introduce a few notations and definitions. We also discuss basic results concerning the collision probability and birthday bound in min- k distributions.

Let D be a discrete probability distribution on set Y defined by probability mass function $D(y) := \Pr_{z \leftarrow D}[z = y]$. The support of D is $\text{Supp}(D) := \{y \in Y : D(y) > 0\}$. We denote $Y^X := \{f : X \rightarrow Y\}$ the set of functions for some domain X and codomain Y . The notation $f \leftarrow Y^X$ indicates that f is a function sampled uniformly from Y^X .

Definition 1 (Min-Entropy). *Let D be a distribution on set Y . D is said to have min-entropy k if $k = -\log_2(\max_{y \in Y}\{D(y)\})$. We refer to a distribution of min-entropy k as a min- k distribution or simply a k -distribution.*

Definition 2 (Flat- k -Distribution). *We call a k -distribution D on set Y a flat- k -distribution, denoted $D_{k,b}$, if the support S of D has size exactly 2^k . It follows that $\forall y \in S, D(y) = 2^{-k}$.*

Definition 3 (δ - k -Distribution). *We call a k -distribution D on set Y a δ - k -distribution if there is a unique mode element $m \in Y$ such that $\forall y \in Y$*

$$D(y) = \begin{cases} 2^{-k} & \text{if } y = m; \\ \frac{1-2^{-k}}{|Y|-1} & \text{otherwise.} \end{cases}$$

We denote such a distribution $D_{k,\delta}$. It is implicit that $|Y| > 2^k$. The support of D is the entire set Y , and remaining probability mass $1 - 2^{-k}$ is distributed uniformly among all elements in Y other than the mode.

Definition 4 (Function of min-entropy k). Let D be a min- k distribution on set Y . We define D^X to be the distribution on Y^X such that for every $x \in X$, its image is sampled independently according to D . $f \leftarrow D^X$ denotes sampling a function in this way, and we say that f is a function of min-entropy k .

Definition 5 (Collision problem). Let $f \leftarrow D^X$ be a function of min-entropy k . A pair of elements $x_1 \in X$ and $x_2 \in X$ such that $x_1 \neq x_2$ and $f(x_1) = f(x_2)$ is called a collision in f . We refer to the problem of producing such a pair as the collision finding problem in D .

Definition 6 (Quantum oracle access). A quantum oracle \mathcal{O} for some function f implements a unitary transformation: $\sum \alpha_{x,y,z} |x, y, z\rangle \xrightarrow{\mathcal{O}} \sum \alpha_{x,y,z} |x, y + f(x), z\rangle$. An algorithm \mathcal{A} that makes (quantum superposition) queries to \mathcal{O} is said to have quantum oracle access to f , and is denoted \mathcal{A}^f .

2.1 Collision Probability and Non-uniform Birthday Bound

Definition 7. The collision probability of a probability distribution D is defined to be the probability that two independent samples from D are equal. Namely

$$\text{CP}(D) := \Pr_{y_1, y_2 \leftarrow D}[y_1 = y_2] = \sum_{y \in Y} D(y)^2.$$

We call $\beta(D) := \frac{1}{\text{CP}(D)}$ the collision variable of D .

$\beta(D)$ will be an important variable determining the complexity of collision finding. In fact we can derive a birthday bound for collisions in an arbitrary distribution D in terms of $\beta(D)$, analogous to the case of uniform distributions, using a key lemma by Wiener [34].

Lemma 1 ([34, Theorem 3]). Let R_D be the random variable denoting the number of i.i.d. samples from a distribution D until a collision appears for the first time. Let $q \geq 1$ be an integer and $\gamma_q := \frac{q-1}{\sqrt{\beta(D)}}$

$$\Pr(R_D > q) \leq e^{-\gamma_q}(1 + \gamma_q).$$

Corollary 1. Let y_1, \dots, y_q be i.i.d. samples from D , and let $\text{COL}^q(D)$ be the event that $y_i = y_j$ for some $i, j \in [q]$. There is a constant $c > 2$ such that if $q \geq c\sqrt{\beta(D)}$, then $\Pr(\text{COL}^q(D)) \geq 2/3$.

Proof. Let E be the event that $y_i = y_j$ for some $i, j \in [q]$. Then

$$\Pr[E] \geq 1 - \Pr[X_D > q] \geq 1 - e^{-\gamma_q}(1 + \gamma_q) \geq 2/3,$$

when $q \geq c\sqrt{\beta(D)}$ because $\frac{1+\gamma_q}{e^{\gamma_q}} < 0.3$ whenever $\gamma_q = \frac{q-1}{\sqrt{\beta(D)}} > 2$.

We can also derive an upper bound on $\Pr[\text{COL}^q(D)]$ by standard approach.

Lemma 2. $\Pr[\text{COL}^q(D)] \leq \frac{q^2}{\beta(D)}$.

Proof. For any pair $i \in [q]$ and $j \in [q]$, Let COL_{ij} be the event that $y_i = y_j$. Then $\Pr[\text{COL}_{ij}] = \text{CP}(D)$. Therefore by union bound, we have

$$\Pr[\text{COL}^q(D)] = \Pr[\cup_{i,j \in [q]} \text{COL}_{ij}] \leq \binom{q}{2} \cdot \text{CP}(D) \leq \frac{q^2}{\beta(D)}.$$

As a result, when $q = o(\sqrt{\beta(D)})$, essentially no collision will occur. Namely q needs to be $\Omega(\sqrt{\beta(D)})$ to see a collision, which is also sufficient by Corollary 1. This is summarized below as a birthday bound for general distributions.

Theorem 1. $\Theta(\sqrt{\beta(D)})$ samples according to D are sufficient and necessary to produce a collision with constant probability for any classical algorithms.

Finally, we characterize $\beta(D)$ for min- k distributions.

Lemma 3. Let D_k be a min- k distribution on Y with $|Y| = N \geq 2^k$ and $k \geq 1$.

- For a flat- k distribution $D_{k,b}$, $\beta(D_{k,b}) = 2^k$.
- For δ -min- k distribution $D_{k,\delta}$, $\beta(D_{k,\delta}) \approx \begin{cases} N & \text{if } N < 2^{2k}; \\ 2^{2k} & \text{if } N \geq 2^{2k}. \end{cases}$
- For a general min- k distribution D_k , $\beta(D_k) \in [2^k, 2^{2k}]$.

Proof. For flat- k D_k , $D_k(y) = \frac{1}{2^k}$ for all $y \in Y' \subseteq Y$ with $|Y'| = 2^k$. Hence $\beta(D_k) = \frac{1}{\sum_{y \in Y'} 2^{-2k}} = 2^k$. For $D_{k,\delta}$ distribution

$$\beta(D_{k,\delta}) = \frac{1}{\text{CP}(D_{k,\delta})} = \frac{1}{2^{-2k} + \frac{(1-2^{-k})^2}{N-1}} = \frac{2^{2k}(N-1)}{N - 2 \cdot 2^k + 2^{2k}} \approx \frac{2^{2k} \cdot N}{2^{2k} + N}.$$

Different ranges of N give the estimation for $\beta(D_{k,\delta})$. For general D_k , it is easy to see that $2^{-2k} \leq \text{CP}(D_k) \leq 2^{-k}$ and hence $\beta(D_k) \in [2^k, 2^{2k}]$.

3 Lower Bounds: Finding a Collision is Difficult

We prove our quantum query lower bounds for min- k collision finding by security reductions. Recall the hardness result for uniform distributions by Zhandry [37].

Lemma 4 ([37] Theorem 3.1). Let $f : [M] \rightarrow [N]$ be a uniformly random function. Then any algorithm making q quantum queries to f outputs a collision in f with probability at most $C(q+1)^3/N$ for some universal constant C .

We show that collision finding in any min- k distribution is at least as difficult as collision finding in a uniform distribution on a set of size 2^k . We begin by demonstrating a reduction of collision finding in a uniform distribution to collision finding in a flat- k distribution. Then we show a reduction of collision finding in a flat- k distribution to collision finding in a general k -distribution. Therefore we prove the following results.

Theorem 2. *Let $f_{flat} \leftarrow D_{k,b}^X$ be a random function whose outputs are chosen independently according to a flat- k -distribution $D_{k,b}$. Then any quantum algorithm making q queries to f_{flat} outputs a collision with probability at most $O((q + 1)^3/2^k)$.*

Theorem 3. *Let $f_D \leftarrow D^X$ be a random function whose outputs are chosen independently according to a distribution D of min-entropy k . Then any quantum algorithm making q queries to f_D outputs a collision with probability at most $O((q + 1)^3/2^k)$.*

Corollary 2. *Any quantum algorithm needs at least $\Omega(2^{k/3})$ queries to find a collision with constant probability in a random function $f_D \leftarrow D^X$ whose outputs are chosen according to a distribution D of min-entropy k .*

Each of the proofs describe an algorithm (i.e., a reduction) attempting to find a collision in a random function f to which it has oracle access. The reduction will run, as a subroutine, another algorithm which finds a collision in another random function g when given oracle access to g (these random functions are not necessarily sampled from the same distribution). To adopt the subroutine which finds collisions in g for the task of finding a collision in f , the reduction simulates an oracle for g by building an oracle converter from the oracle for f and a suitable redistribution function. In general the redistribution function must be random, sampled from a particular distribution so that the distribution of its images equals that of g . Given some distributions from which the images of f and g are sampled, only some special sampling procedures will produce a redistribution function suitable for building the oracle converter needed. We formalize the concept of a *redistribution function sampler* as a generally randomized algorithm that performs such a sampling procedure specific to the oracles the reduction has access to and needs to simulate.

Definition 8 ($D \rightarrow D'$ Redistribution Function Sampler). *Suppose $f : X \rightarrow Y$ is a random function whose images are distributed according to a distribution D . Let D' be a distribution on Y' . We call an algorithm S a $D \rightarrow D'$ redistribution function sampler if it returns a function $r : X \times Y \rightarrow Y'$ such that $\Pr[r(x, f(x)) = y] = D'(y)$ for all $y \in Y'$ and $x \in X$, where the probability is taken over the random choice of f and the randomness of S .*

We use the term *redistribution function* to refer to a function returned by a redistribution function sampler, explicitly stating the distributions when necessary. The redistribution function naturally induces an oracle converter.

Definition 9 (Oracle Converter). *Suppose $f \leftarrow D^X$ is a random function whose images are distributed according to a distribution D on Y . Let D' be a distribution on Y' , and $r : X \times Y \rightarrow Y'$ be a $D \rightarrow D'$ redistribution function. An algorithm C , having oracle access to f and r , is called an oracle converter from f to g if C computes a function $g : X \rightarrow Y'$ defined by $g(x) := r(x, f(x))$.*

We may denote $g = C_{f,r}$. We can immediately observe that g is distributed as if the images were sampled independently according to D' , when f and r are sampled according to the above definition.

Lemma 5. *The oracle converter defined above computes a function g that is distributed identically to D'^X , i.e., its images are independently distributed according to D' , if $f \leftarrow D^X$ is chosen randomly and r is generated by a $D \rightarrow D'$ redistribution function sampler.*

We will be concerned with finding collisions in f and g . In particular, we are interested in whether a collision of g constitutes a collision of f . We define the collision-conversion rate to capture this property of an oracle converter.

Definition 10 (Collision-conversion rate). *Let \mathcal{C} be an oracle converter from f to g . We say that it has collision-conversion rate p if for any (x, x') such that $g(x) = g(x')$, $f(x) = f(x')$ also holds with probability at least p . The probability is over the random choices of $f \leftarrow D^X$ and of a $D \rightarrow D'$ redistribution function r .*

With these notions available, our reduction proofs basically sample a proper redistribution function, and then simulate a correct oracle g distributed according to D'^X using an oracle converter accessing the given oracle $f \sim D^X$. Then we run a collision-finding adversary on D' with oracle g . Whenever it outputs a collision, we can conclude that a collision is also found in f with probability p by the collision-conversion rate, which will lead to the desired contradiction. For each of the reductions, we will describe a suitable redistribution function sampler and show that it has at least constant collision-conversion rate. To do so, we assume that the reductions have full information about D and D' , as well as sufficient randomness. This is fine as far as query complexity is concerned, and it is an interesting open question to make them time-efficient. We also remark that, for the sake of clarity, the distribution of images of our redistribution function is defined to be *exactly* matching distribution D' . It suffices to approximate distribution D' up to some negligible statistical distance.

Now we provide a generic formal description for all of our reductions, leaving the redistribution function sampler as a modular component which we can describe individually for each collision finding problem (for now we assume that each reduction has access to an adequate redistribution function sampler in each case). We do this in part to formally demonstrate how our reductions are compatible with *quantum* adversaries, allowing them to submit queries in quantum superposition and receive the oracle responses in quantum superposition. We will show that the oracle converters can be implemented as quantum oracles, so that the reduction can simulate the collision-finding problem for a quantum adversary who submit quantum queries. As usual, we consider a reduction solving collision-finding in D using an adversary for collision-finding in D' .

We emphasize that the functions f and r are random functions sampled before the adversary begins the attack (the *attack* referring to the query-response phase in which interaction with the oracle occurs), as f is simply a model for what would be a fixed, publicly known hash function in a practical security setting, and r would be chosen by the adversary according to some procedure specific to the hash function (this is the role played by redistribution function sampler). Implementing the converter as a quantum-accessible oracle is straightforward

Algorithm 1. Generic reduction via oracle converter

Input: Let $f \leftarrow D^X$ be a random function whose images are sampled according to D on a set Y . Let D' be a distribution on a set Y' . Let S be a $D \rightarrow D'$ redistribution function sampler. Let \mathcal{A} be an adversary for collision-finding in D' .

Output: A possible collision (x_1, x_2) in f .

- 1: Run S and store its output as r . Implement an oracle for r .
 - 2: Construct an oracle converter \mathcal{C} using the oracles for f and r . The responses of \mathcal{C} are now distributed according to D' . Refer to the function implemented by \mathcal{C} as g .
 - 3: Initialize \mathcal{A} . For each query made by \mathcal{A} , forward the query to \mathcal{C} and return the response to \mathcal{A} .
 - 4: When \mathcal{A} returns a collision (x_1, x_2) in g , output (x_1, x_2) .
-

as shown below (Fig. 1). Note that the function r can be turned into a unitary operator by standard technique $|x, \tilde{x}, y\rangle \xrightarrow{r} |x, \tilde{x}, y \oplus r(x, \tilde{x})\rangle$. f is given as a quantum oracle, which we just need to query twice to answer each query to g .

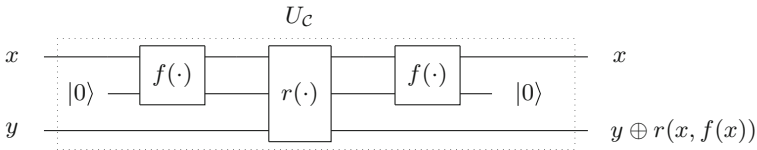


Fig. 1. Quantum circuit that implements function $g = \mathcal{C}_{f,r}$ using two oracle calls to f .

Now that we have a generic construction for our reductions, we will show a simple reusable general result that will allow us to quickly construct reductions and extend query complexity lower bounds by simply demonstrating the existence of a satisfactory redistribution function sampler for use in each reduction. In this context we say that a reduction algorithm *succeeds* if the output pair indeed forms a collision in the given oracle function.

Lemma 6. *Suppose there exists an algorithm \mathcal{A} which solves collision finding in a distribution D' with probability at least P_A , using q queries to an oracle for a function g whose responses are distributed according to D' ¹. Suppose there exists a $D \rightarrow D'$ redistribution function sampler S such that the induced converter has collision-conversion rate at least p . Then Algorithm 1 initialized with S and \mathcal{A} , denoted $\mathcal{R}_{S,A}$, solves collision finding in D with probability at least $p \cdot P_A$ using $2q$ queries to an oracle for f whose images are distributed according to D .*

Proof. Lemma 6 follows trivially from the suppositions stated. Let A denote the event that \mathcal{A} succeeds, E denote the event that the a collision of g is also a collision of f , and R denote the event that $\mathcal{R}_{S,A}$ succeeds. Then

$$\Pr[R] \geq \Pr[E \cap A] = \Pr[E|A] \cdot \Pr[A] = \Pr[E] \cdot \Pr[A],$$

¹ The probability P_A reflects the randomness of oracle’s responses and of \mathcal{A} .

because E and A are independent ($\Pr[E]$ is simply the collision-conversion rate, which is a property specific to the oracle converter used in Algorithm 1). Since $\Pr[E] \geq p$ and $\Pr[A] \geq P_A$, $\Pr[R] \geq p \cdot P_A$. The observation that $\mathcal{R}_{S,A}$ uses twice the number of oracle queries as \mathcal{A} proves the lemma.

Therefore to prove Theorems 2 and 3, all that is left is to show suitable redistribution function samplers.

Lemma 7. *Let U_{2^k} be a uniform distribution on a set Y of size 2^k . Let $D_{k,b}$ be a flat- k distribution on a set Y_1 , and D_k a general min- k distribution on a set Y_2 . There exist $U_{2^k} \rightarrow D_{k,b}$ and $D_{k,b} \rightarrow D_k$ redistribution function samplers, and the induced oracle converters have collision-conversion rates at least $1/2$.*

Proof. We describe the two samplers below.

$U_{2^k} \rightarrow D_{k,b}$ sampler. In this case the redistribution function sampler is nearly trivial because a simple relabeling of samples from the distribution U_{2^k} will suffice to simulate samples from the distribution $D_{k,b}$. Let f be a function $f : X \rightarrow Y$ whose images are distributed according to U_{2^k} , to which oracle access is available. Let $m : Y \rightarrow Y_1$ be any injective mapping. Define S_1 as a one-step algorithm that returns a function $r_1(x, y) = m(y)$.

By the definition of r_1 , $\Pr[r_1(x, f(x)) = y'] = \Pr[m(f(x)) = y']$ for all $x \in X$ and $y' \in Y_1$. Since m implements an injective mapping from Y to Y_1 , $\Pr[m(f(x)) = y'] = \Pr[f(x) = m^{-1}(y')]$. Since, by the definition of f , $\Pr[f(x) = y] = U_{2^k}(y)$ for all $y \in Y$, $\Pr[f(x) = m^{-1}(y')] = U_{2^k}(m^{-1}(y')) = 2^{-k}$. Hence $\Pr[r_1(x, f(x)) = y'] = D_{k,b}(y')$ for all $x \in X$ and $y' \in Y_1$, since $D_{k,b}(y') = 2^{-k}$ for all $y' \in Y_1$. It follows that S_1 is a $U_{2^k} \rightarrow D_{k,b}$ redistribution function sampler. We now show that the collision-conversion rate of the induced oracle converter is exactly 1. Let (x_1, x_2) be a collision in g , the function implemented by the oracle converter. Then $r_1(x_1, f(x_1)) = r_1(x_2, f(x_2))$, from which it follows that $m(f(x_1)) = m(f(x_2))$. Since m is an injective mapping, we can conclude that $f(x_1) = f(x_2)$, which shows that (x_1, x_2) is necessarily a collision in f .

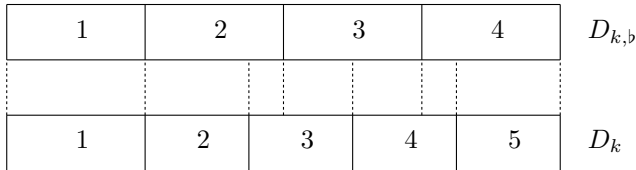
$D_{k,b} \rightarrow D_k$ sampler. We provide an overview of the $D_{k,b} \rightarrow D_k$ redistribution function sampler in the following few paragraphs. The complete redistribution function sampler is given in the full version, along with a detailed explanation of the reasoning behind it. We reiterate that the redistribution function can be prepared before oracle access to the hash function under attack is obtained, allowing the query-response phase of the attack to be implemented as a quantum algorithm without concern for the quantum implementation of the redistribution function sampler.

The basic challenge that must be solved by the redistribution function sampler is to provide a mapping from the support of one distribution to the support of another distribution in such a way that the output is actually distributed according to the second distribution, which we call D_k , when the input is

distributed according to the first, which we call $D_{k,b}$ ². In order to maximize the probability that Algorithm 1 succeeds, the mapping must maximize the probability that two identical outputs correspond with two identical inputs, i.e., the collision-conversion rate. Our construction for this redistribution function sampler, which we call S_2 (and which returns a function which we call r_2), ensures that this probability is no less than one half by allowing at most two elements of the support of the $D_{k,b}$ be mapped to each element of the support of D_k . To provide intuition for how this is achieved, we recommend visualizing each distribution as a rectangle divided into ‘bins’ representing the elements of its support, with each bin’s width proportional to the probability mass of the corresponding element under the distribution. We refer to this as the *rectangular representation* of the distribution. An example is shown below. We let $D_{k,b}$ be a flat distribution of min-entropy 2, and D_k be a (non-flat) distribution of min-entropy 2. We label each bin with a number indexing the elements of the support in each case.



For each of the elements of the support of $D_{k,b}$, we must decide what the probability mass corresponding to that element in $D_{k,b}$ should ‘be sent to’ by the redistribution function, in the sense that whatever that element is mapped to will occur with the same probability as that of sampling the element from $D_{k,b}$. A natural solution that would correctly produce the distribution D_k is to in some sense ‘project’ the distribution $D_{k,b}$ onto D_k , so that each ‘location’ in the rectangular representation of $D_{k,b}$ is mapped to a ‘location’ in the rectangular representation of D_k (by ‘location’ here we refer to horizontal position a rectangular representation, selecting some specific probability density). We illustrate this sort of projection by drawings lines between the two rectangular representations that show where the boundaries between the elements of each distribution’s support fall in the other distribution, shown below.



² A redistribution function formally is also provided the query x that is associated with the sample from the first distribution, which is (in Algorithm 1) the response from an oracle whose output is distributed according to the first distribution. This is necessary in cases where the second distribution has a larger support than the first, since the image of the redistribution function cannot be larger than the domain. It can safely be ignored otherwise (as in the construction for r_1).

From the fact that the width of each bin is proportional to the probability mass associated with each element of each distribution, it follows that, if, for a given sample from $D_{k,b}$, we sample an element from the support of D_k according to the available probability mass *inside* the projected bin from $D_{k,b}$, the sampling result will be distributed exactly according to the distribution D_k . This is difficult to communicate verbally, but visually, one can imagine receiving a sample from $D_{k,b}$ as ‘selecting’ the bin associated with the sampled value in the rectangular representation of the distribution. Then, following the lines bordering that bin, we find that the probability mass associated with the sample from $D_{k,b}$ is mapped to probability mass corresponding to several elements of the support of distribution D_k . If we now sample from these elements according to their share of the probability mass corresponding to the sample from $D_{k,b}$, our samples will be distributed according to D_k . For example, with reference specifically to the graphic above, suppose that we receive element 2 as a sample from $D_{k,b}$. Following the lines down from the bin corresponding to element 2 in the rectangular representation of $D_{k,b}$, we see that elements 2 and 3 in the support of D_k both partially reside in the space corresponding to bin 2 in the rectangular representation of $D_{k,b}$. In particular, element 2 in the support of D_k consumes much more of the space than element 3. Hence we sample either 2 or 3, with a bias toward 2 exactly equal to how much more of the space element 2 consumes (recall that *space* in these rectangular representations corresponds to probability mass). Similarly, had we received element 3 as a sample from $D_{k,b}$, we would have sampled from elements 3 and 4 in the support of D_k with little or no bias, since these seem to roughly evenly split the space inside the boundaries of the bin corresponding to element 3 in the support of $D_{k,b}$.

It should be clear now that this procedure will produce samples distributed according to D_k when given samples distributed according to $D_{k,b}$, at the cost of needing additional randomness to perform the sub-sampling. Generating the redistribution function r_2 is then simply a matter of saving the resulting samples in a look-up table. Although this procedure is conceptually simple, its rigorous mathematical description is exceedingly tedious, so we provide it in the full version of this paper. Also in the full version is a proof that the redistribution function sampler S_2 has a collision-conversion rate of at least one-half. The intuition behind this property is that a sample from D_k produced by the redistribution function could have been generated by, at most, 2 distinct samples from $D_{k,b}$, since each bin in the rectangular representation of D_k resides within the boundaries of, at most, 2 bins in the rectangular representation of $D_{k,b}$.

We have shown that S_1 and S_2 , as just described (and formally described in the full version in the case of S_2), are $U_{2^k} \rightarrow D_{k,b}$ and $D_{k,b} \rightarrow D_k$ redistribution function samplers, respectively. Finally, Theorems 2 and 3 follow easily. Note that we write some of the constant factors in the probabilities with the common notation C , even though they will not all take the same numerical value, in recognition that they are not interesting for the study of asymptotic query complexity.

Proof (Proof of Theorems 2 and 3). By Lemma 7, there exists a $U_{2^k} \rightarrow D_{k,b}$ redistribution function sampler S_1 for which the induced collision-conversion rate is at least one-half. Therefore Lemma 6 implies that our reduction algorithm is an collision-finding adversary making $2q$ queries to a uniformly random function f with success probability at least $P_{\mathcal{A}}/2$. However, Lemma 4 tells us that any $2q$ -query adversary can succeed with probability at most $C(2q + 1)^3/2^k$. Therefore the success probability $P_{\mathcal{A}}$ of any q -query adversary \mathcal{A} is $O(q + 1)^3/2^k$, which proves Theorem 2.

Theorem 3 is proved in the same fashion by invoking the $D_{k,b} \rightarrow D_k$ redistribution function sampler S_2 in Lemma 7 and with Theorem 2 taking the place of Lemma 4.

3.1 Stronger Lower Bound for δ -min- k Distributions

Note that following the same strategy, one can show a reduction of collision finding in an arbitrary min- k distribution D to collision finding in a δ - k -distribution. This is interesting because it affirms that the δ - k -distribution case is the most difficult out of all k -distributions. Clearly, if no elements in the support of D are associated with a probability mass less than $1/N$, the proof of Theorem 3 can be adapted by replacing all references of 2^{-k} as the probability of sampling each element from the flat distribution with a general probability $D(x)$, and replacing the general distribution D with a δ - k -distribution D_{δ} . The general case where D has elements associated with smaller probability mass than $1/N$ may be resolved by considering the distribution removing these elements and showing that it is computationally indistinguishable from the original.

In this section we give further evidence and establish an even stronger bound for finding collision in the δ - k -distribution case.

Theorem 4. *For any q -query algorithm A ,*

$$\Pr_{f \leftarrow D_{k,\delta}^X} [f(x) = f(x') : (x, x') \leftarrow A^f(\cdot)] \leq O\left(\frac{(q + 2)^2}{2^k} + \frac{(q + 2)^3}{N}\right).$$

We give two proofs. The one presented here relies on a technique by Zhandry (Lemma 8). We give an alternative proof in the full version based on a reduction from an average version of a search problem which is hard to solve from the literature. This may serve as an intuitive explanation of the hardness of non-uniform collision finding. It also connects to the quantum algorithm we develop in Sect. 4.1 based on Grover’s search algorithm.

Lemma 8 [36, Theorem 7.2]. *Fix q , and let F_{λ} be a family of distributions on Y^X indexed by $\lambda \in [0, 1]$. Suppose there is an integer d such that for every $2q$ pairs $(x_i, y_i) \in X \times Y$, the function $p_{\lambda} := \Pr_{f \leftarrow F_{\lambda}}(f(x_i) = y_i, \forall i \in \{1, \dots, 2q\})$ is a polynomial of degree at most d in λ . Then any quantum algorithm A making q queries can only distinguish F_{λ} from F_0 with probability at most $2\lambda d^2$.*

This lemma enables us to prove the following proposition.

Proposition 1. For any q -query algorithm A ,

$$\left| \Pr_{f \leftarrow D_{k,\delta}^X}(A^f(\cdot) = 1) - \Pr_{f \leftarrow Y^X}(A^f(\cdot) = 1) \right| \leq 8q^2/2^k + 1/N.$$

Proof. For every $\lambda \in [0, 1]$, define D_λ on Y such that there is an element $m \in Y$ with $D_\lambda(m) = \lambda$ and for any $y \neq m$ $D_\lambda(y) = \frac{1-\lambda}{|Y|-1}$. Then define a family of distributions F_λ on Y^X where $F_\lambda := D_\lambda^X$, i.e., the output of each input is chosen independently according to D_λ .

For any $\{(x_i, y_i)\}_{i=1}^{2q}$, $p_\lambda := \Pr_{f \leftarrow F_\lambda}(f(x_i) = y_i, \forall i \in [2q]) = \lambda^t (\frac{1-\lambda}{|Y|-1})^{2q-t}$, where t is the number of occurrences of m in $\{y_i\}_{i=1}^{2q}$. Clearly p_λ is a polynomial in λ with degree at most $2q$.

Notice that $F_{2^{-k}}$ is exactly δ -min- k distribution $D_{k,\delta}$, and F_0 is uniformly random on \hat{Y}^X , where $\hat{Y} := Y \setminus \{m\}$. Therefore by Lemma 8,

$$\left| \Pr_{f \leftarrow D_{k,\delta}^X}(A^f(\cdot) = 1) - \Pr_{f \leftarrow \hat{Y}^X}(A^f(\cdot) = 1) \right| \leq 2(2q)^2 \cdot 2^{-k} = 8q^2/2^k.$$

Since Y^X and \hat{Y}^X has statistical distance $\frac{1}{2}(N-1)(\frac{1}{N-1} - \frac{1}{N}) + \frac{1}{2}(\frac{1}{N} - 0) = 1/N$, we get that $\left| \Pr_{f \leftarrow D_{k,\delta}^X}(A^f(\cdot) = 1) - \Pr_{f \leftarrow Y^X}(A^f(\cdot) = 1) \right| \leq 8q^2/2^k + 1/N$.

We are now ready to prove the stronger complexity for finding collision in a δ -min- k random function.

Proof (Proof of Theorem 4). Suppose that there is an A with

$$\Pr_{f \leftarrow D_{k,\delta}^X}[f(x) = f(x') : (x, x') \leftarrow A^f(\cdot)] = \varepsilon$$

using q queries. Then construct A' which on input oracle f , runs A and receives (x, x') from A . A' then output 1 iff. $f(x) = f(x')$. By definition, we have that $\Pr_{f \leftarrow D_{k,\delta}^X}(A'^f(\cdot) = 1) = \varepsilon$. Meanwhile, note that A' makes $q+2$ queries. Therefore by Zhandry's lower bound on finding collision in uniform random function (Lemma 4), we know that $\Pr_{f \leftarrow Y^X}(A'^f(\cdot) = 1) \leq O(\frac{(q+3)^3}{N})$. Then Proposition 1 implies that

$$\varepsilon \leq O\left(\frac{(q+3)^3}{N}\right) + 8(q+2)^2/2^k + 1/N = O\left(\frac{(q+2)^2}{2^k} + \frac{(q+3)^3}{N}\right).$$

Corollary 3. Any quantum algorithm needs $\min\{2^{k/2}, N^{1/3}\}$ queries to find a collision with constant probability. Specifically we need $\Omega(N^{1/3})$ if $2^k \leq N < 2^{\frac{3k}{2}}$, and $\Omega(2^{k/2})$ when $N \geq 2^{\frac{3k}{2}}$.

4 Upper Bounds: (Optimal) Quantum Algorithms

We derive a generic upper bound for finding collision in any min- k random functions. We adapt Ambainis’s algorithm (Lemma 9) and describe a quantum algorithm NU-CoIF (Algorithm 2).

Lemma 9 ([8, Theorem 3]). *Let $f : X' \rightarrow Y$ be a function that has at least one collision. Then there is a quantum algorithm CoIF making $O(|X'|^{2/3})$ quantum queries to f that finds the collision with constant bounded error.*

Algorithm 2. Collision Finding in Non-uniform Function NU-CoIF

Input: $f \leftarrow D_k^X$ as an oracle. Let s, t be parameters to be specified later.

Output: Collision (x, x') or \perp .

- 1: Divide X in to subsets X_i of equal size (ignoring the boundary case) $|X_i| = s$.
 - 2: Construct $f_i : X_i \rightarrow Y$ as the restriction of f on X_i .
 - 3: For $i = 1, \dots, t$, Run Ambainis’s algorithm CoIF on f_i , and get candidate collision x_i and x'_i . if $f(x_i) = f(x'_i)$, output (x_i, x'_i) and abort.
 - 4: Output \perp .
-

Theorem 5. *Let $\beta := \beta(D_k)$. Let X be a set with $|X| = M = \Omega(\sqrt{\beta})$. Algorithm 2 NU-CoIF finds a collision in $f \leftarrow X^{D_k}$ within $O(\beta^{1/3})$ queries with constant probability. Moreover with $O(k\beta^{1/3})$ queries the algorithm succeeds except with probability negligible in k .*

Proof. Since f is generated according to the min- k distribution, when restricting to any subset X_i , we can think of drawing each function value independently from D_k . Namely $f_i \sim D_k^{X_i}$ holds for all i . Therefore, by Lemma 1, we have that when $s \geq c\sqrt{\beta(D)}$ for some $c > 2$, f_i contains a collision with constant probability. If that is the case, Ambainis’s algorithm will find a collision with constant probability using $O(|X_i|^{2/3}) = O(\beta(D)^{1/3})$ queries. We only need to repeat $t = O(k)$ times to succeed except with error negligible in k .

Note that our algorithm NU-CoIF is generic, and needs no additional information about D_k . By our characterization of $\beta(D_k)$ in Lemma 3, we obtain specific bounds for the two special distributions.

Corollary 4. *There exists a quantum algorithm that finds a collision with constant probability using the following numbers of queries:*

- flat- k : $O(\beta^{1/3}) = O(2^{k/3})$ and it is tight when $M = \Omega(2^{k/2})$.
- δ -min- k : $O(\beta^{1/3}) = \begin{cases} O(N^{1/3}) & 2^k \leq N < 2^{2k}, \text{ tight when } N \leq 2^{3k/2} \\ O(2^{\frac{2k}{3}}) & N \geq 2^{2k}. \end{cases}$

4.1 Quantum Algorithm for min- k Distribution with a Mode Known

We design an alternative collision finding algorithm (Algorithm 3), which performs slightly better in some settings. It is based on a version of Grover’s algorithm [15, 22, 28] for multiple marked items stated below.

Lemma 10. *Let $f : X \rightarrow \{0, 1\}$ be an oracle function and let $Z_f = |\{x \in X : f(x) = 1\}|$. Then there is a quantum algorithm QSEARCH using q queries that finds an $x \in X$ such that $f(x) = 1$ with success probability $\Omega(q^2 \frac{Z_f}{|X|})$.*

Algorithm 3. Collision Finding in Non-uniform Function with a mode known NU-ColF-Mode

Input: $f \leftarrow D_k^X$ as an oracle. A mode element m of D_k .

Output: Collision (x, x') or \perp .

- 1: Run Grover’s algorithm QSEARCH on f to find x with $f(x) = m$.
 - 2: Run Grover’s algorithm QSEARCH on f to find x' with $f(x) = m$ and $x' \neq x$.
 - 3: Output \perp if any run of the Grover’s algorithm failed. Otherwise output (x, x') .
-

Theorem 6. *NU-ColF-Mode finds a collision using $O(2^{k/2})$ queries with constant probability.*

Proof. Let $Z_f := |f^{-1}(m)|$. Let p_f be the probability that f is chosen, when drawn from D_k^X . Since we invoke QSEARCH twice, we find (x, x') with probability $\Omega\left(\left(\frac{q^2 Z_f}{|X|}\right)^2\right)$. Then algorithm NU-ColF-Mode succeeds with probability

$$\sum_f p_f \Omega\left(\frac{q^4}{M^2} Z_f^2\right) = \Omega\left(\frac{q^4}{M^2} \sum_f p_f Z_f^2\right) = \Omega\left(\frac{q^4}{M^2} \mathbb{E}[Z_f^2]\right).$$

To compute $\mathbb{E}[Z_f^2]$, we define for every $x \in X$ an indicator variable $Z_x = \begin{cases} 1 & \text{if } f(x) = m; \\ 0 & \text{otherwise.} \end{cases}$, where $f \leftarrow D_k^X$, and clearly $Z_f = \sum_{x \in X} Z_x$. Since each output of x is drawn independently according to $D_{k,\delta}$, $\mathbb{E}[Z_x] = \varepsilon := 2^{-k}$ for all x , it follows that $\mathbb{E}[Z_x] = \mathbb{E}[Z_x^2] = \varepsilon$, and $\mathbb{E}[Z_x \cdot Z_{x'}] = \mathbb{E}[Z_x] \cdot \mathbb{E}[Z_{x'}] = \varepsilon^2$ for any $x \neq x'$ by independence. Therefore

$$\mathbb{E}[Z_f^2] = \sum_x \mathbb{E}[Z_x^2] + \sum_{x \neq x'} \mathbb{E}[Z_x Z_{x'}] = \Omega(M^2 \varepsilon^2).$$

Hence the algorithm succeeds with probability $\Omega(q^4 \varepsilon^2) = \Omega\left(\frac{q^2}{2^k}\right)^2$. As a result, with $q = O(2^{k/2})$ many queries, we find a collision with constant probability.

Remark 1. Note that we still need $M = \Omega(\sqrt{\beta(D)})$ to ensure existence of collisions. When $N \geq 2^{3k/2}$, Theorem 6 gives a better bound ($2^{k/2}$) than Theorem 5 ($N^{1/3}$ when $2^{3k/2} \leq N < 2^{2k}$ and $2^{2k/3}$ when $N \geq 2^{2k}$).




References

1. Password hashing competition (2012). <https://password-hashing.net/>
2. National Institute of Standards and Technology. SHA-3 standard: permutation-based hash and extendable-output functions (2014). http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf
3. IBM Q quantum experience (2017). <https://www.research.ibm.com/ibm-q/>
4. National Institute of Standards and Technology. FIPS 180–1: secure hash standard, April 1995
5. People of ACM - John Martinis, 16 May 2017. <https://www.acm.org/articles/people-of-acm/2017/john-martinis>
6. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. *J. ACM (JACM)* **51**(4), 595–605 (2004)
7. Ambainis, A.: Polynomial degree and lower bounds in quantum complexity: collision and element distinctness with small range. *Theory Comput.* **1**(3), 37–46 (2005). <http://www.theoryofcomputing.org/articles/v001a003>
8. Ambainis, A.: Quantum walk algorithm for element distinctness. *SIAM J. Comput.* **37**(1), 210–239 (2007). Preliminary version in FOCS 2004. [arXiv:quant-ph/0311001](https://arxiv.org/abs/quant-ph/0311001)
9. Ambainis, A., Rosmanis, A., Unruh, D.: Quantum attacks on classical proof systems (the hardness of quantum rewinding). In: FOCS 2014, pp. 474–483. IEEE, October 2014. Preprint on IACR ePrint 2014/296
10. Amy, M., Di Matteo, O., Gheorghiu, V., Mosca, M., Parent, A., Schanck, J.: Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. *arXiv preprint arXiv:1603.09383* (2016)
11. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, pp. 62–73. ACM (1993)
12. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995). <https://doi.org/10.1007/BFb0053428>
13. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak sponge function family (2007). <http://keccak.noekeon.org/>
14. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3
15. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. *arXiv preprint arXiv:quant-ph/9605034* (1996)
16. Brassard, G., Hoyer, P., Tapp, A.: Quantum algorithm for the collision problem. *arXiv preprint arXiv:quant-ph/9705002* (1997)
17. Crépeau, C., Salvail, L., Simard, J.-R., Tapp, A.: Two provers in isolation. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 407–430. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_22
18. Czajkowski, J., Bruinderink, L.G., Hülsing, A., Schaffner, C., Unruh, D.: Post-quantum security of the sponge construction. *Cryptography ePrint Archive, Report 2017/771* (2017). <https://eprint.iacr.org/2017/771>
19. Eaton, E., Song, F.: Making existential-unforgeable signatures strongly unforgeable in the quantum random-oracle model. In: 10th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2015. LIPIcs, vol. 44, pp. 147–162. Schloss Dagstuhl (2015)

20. Ebrahimi, E., Unruh, D.: Quantum collision-resistance of non-uniformly distributed functions: upper and lower bounds. Cryptology ePrint Archive, Report 2017/575 (2017)
21. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptol.* **26**(1), 80–101 (2013). Preliminary version in CRYPTO 1999
22. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 212–219. ACM (1996)
23. Hallgren, S., Smith, A., Song, F.: Classical cryptographic protocols in a quantum world. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 411–428. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_23
24. Hülsing, A., Rijneveld, J., Song, F.: Mitigating multi-target attacks in hash-based signatures. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 387–416. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_15
25. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). <https://bitcoin.org/bitcoin.pdf>
26. Rivest, R.L.: RFC 1321: the MD5 message-digest algorithm, April 1992. <https://www.ietf.org/rfc/rfc1321.txt>
27. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**(5), 1484–1509 (1997)
28. Song, F.: Early days following Grover’s quantum search algorithm. arXiv preprint [arXiv:1709.01236](https://arxiv.org/abs/1709.01236) (2017)
29. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. Cryptology ePrint Archive, Report 2017/190 (2017). <https://shattered.io/>
30. Targhi, E.E., Tabia, G.N., Unruh, D.: Quantum collision-resistance of non-uniformly distributed functions. In: Takagi, T. (ed.) PQCrypto 2016. LNCS, vol. 9606, pp. 79–85. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29360-8_6
31. Targhi, E.E., Unruh, D.: Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 192–216. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_8
32. Unruh, D.: Computationally binding quantum commitments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 497–527. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_18
33. Watrous, J.: Zero-knowledge against quantum attacks. *SIAM J. Comput.* **39**(1), 25–58 (2009). Preliminary version in STOC 2006
34. Wiener, M.J.: Bounds on birthday attack times. Cryptology ePrint Archive, Report 2005/318 (2005). <http://eprint.iacr.org/2005/318>
35. Yuen, H.: A quantum lower bound for distinguishing random functions from random permutations. *Quantum Inf. Comput.* **14**(13–14), 1089–1097 (2014)
36. Zhandry, M.: How to construct quantum random functions. In: FOCS 2012, pp. 679–687. IEEE (2012). <http://eprint.iacr.org/2012/182>
37. Zhandry, M.: A note on the quantum collision and set equality problems. *Quantum Inf. Comput.* **15**(7 & 8), 557–567 (2015)
38. Zhandry, M.: Secure identity-based encryption in the quantum random oracle model. *Int. J. Quantum Inf.* **13**(4) (2015). Early version in Crypto 2012. <http://eprint.iacr.org/2012/076>



Asymptotically Faster Quantum Algorithms to Solve Multivariate Quadratic Equations

Daniel J. Bernstein¹  and Bo-Yin Yang²  

¹ Department of Computer Science, University of Illinois at Chicago,
Chicago, IL 60607-7045, USA

djb@cr.y.p.to

² Institute of Information Science, Academia Sinica,
128 Section 2 Academia Road, Taipei 115-29, Taiwan

by@crypto.tw

Abstract. This paper designs and analyzes a quantum algorithm to solve a system of m quadratic equations in n variables over a finite field \mathbf{F}_q . In the case $m = n$ and $q = 2$, under standard assumptions, the algorithm takes time $2^{(t+o(1))n}$ on a mesh-connected computer of area $2^{(a+o(1))n}$, where $t \approx 0.45743$ and $a \approx 0.01467$. The area-time product has asymptotic exponent $t + a \approx 0.47210$.

For comparison, the area-time product of Grover's algorithm has asymptotic exponent 0.50000. Parallelizing Grover's algorithm to reach asymptotic time exponent 0.45743 requires asymptotic area exponent 0.08514, much larger than 0.01467.

Keywords: FXL · Grover · Reversibility · Bennett–Tompa
Parallelization · Asymptotics

1 Introduction

By definition, a NAND gate reads two bits $a, b \in \mathbf{F}_2$ as input and produces a bit $c = 1 - ab$ as output. It is well known that any function from ℓ -bit strings to ℓ' -bit strings, for any ℓ and any ℓ' , can be viewed as being computed by a circuit built from NANDs.

Author list in alphabetical order; see <https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf>. This work was supported by the European Commission under Contract ICT-645622 PQCRYPTO; by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005; and by the U.S. National Science Foundation under grant 1314919. This work also was supported by Taiwan Ministry of Science and Technology (MoST) grant 105-2923-E-001-003-MY3 and an Academia Sinica Investigator Award. “Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation” (or other funding agencies). Permanent ID of this document: c77423932ceeda61ddf009049efc0749daadd023. Date: 2018.01.25.

For example, one can compute the 2-bit-to-2-bit function $(a, b) \mapsto (ab, a+b)$ by computing $c = 1-ab, d = 1-ac, e = 1-bc, f = 1-cc, g = 1-de$; note that $f = ab$ and $g = a + b$. By further composition one can build, e.g., an integer-addition circuit producing a 32-bit output from two 32-bit inputs; a circuit computing a 256-bit SHA-256 output from a fixed-length input; and a circuit computing a 2048-bit RSA public key as a product of two secret 1024-bit primes.

Circuits built from NANDs are one of the standard models of computation. They are also, fundamentally, how computation is carried out today.¹ The problem of inverting one of today’s computations—for example, finding a preimage for a hash output, or finding a secret key given a public key, or finding a plaintext given a public key and a ciphertext—can thus be viewed as the problem of solving a system of multivariate quadratic (“MQ”) equations. **Quadratic** here means “degree at most 2”, so quadratic equations include linear equations.

Specifically, each NAND gate can be expressed as a quadratic equation in at most three variables, such as the equation $c = 1 - ab$ in variables a, b, c , or the equation $f = 1 - cc$ in two variables c, f . Note that the second equation can be simplified to the linear equation $f = 1 - c$, using the fact that $c^2 = c$. Each known output bit for the computation can be expressed as a linear equation such as $g = 0$.

1.1 Random Systems of MQ Equations. Formally, a quadratic equation $\sum_{j>k} \alpha_{j,k} x_j x_k = \beta$ in n variables x_1, x_2, \dots, x_n over \mathbf{F}_2 is specified by a sequence of $n(n+1)/2+1$ coefficients $\alpha_{1,1}, \alpha_{2,1}, \alpha_{2,2}, \dots, \alpha_{n,n}, \beta$. A system of m quadratic equations in n variables is thus specified by $m(n(n+1)/2+1)$ coefficients. The equation-solving problem is to determine, given these coefficients, whether there exists a solution $(x_1, x_2, \dots, x_n) \in \mathbf{F}_2^n$ to all m equations, and if so to find *some* solution. Note that a reliable method to determine existence of a solution can be used recursively to find a solution.

There is a vast literature on fast equation-solving techniques that rely on special structure of the coefficients. For example, systems of linear equations ($\alpha_{j,k} = 0$ if $j \neq k$) are easy to solve. More generally, any nonzero linear equation can be eliminated, along with one of the variables used in the equation, producing a system of $m - 1$ quadratic equations in $n - 1$ variables. As another example, some systems have a “triangular” structure that makes them easy to solve: one can first solve for one variable without regard to the rest, then solve for another variable, etc. As yet another example, the problem of factoring a 256-bit integer into two 128-bit factors has a tremendous amount of mathematical structure, and this structure is exploited by factorization algorithms that run in mere minutes on a laptop today. But none of these examples have a noticeable chance of applying to a uniform random system with $m = n = 256$: a system of 256

¹ One can object to the circuit model of computation as being too restrictive: (1) in the algorithms literature it is common to treat random access to an arbitrarily large array as a single operation taking a single unit of “time”; (2) the algorithms literature also allows “branches”. However, (1) for any particular size of array, random access can be implemented as a series of NANDs—which is essentially how physical RAM devices work; (2) branches are equivalent to—and physically implemented as—random access to an array of instructions.

equations in 256 variables in which each coefficient is chosen uniformly and independently at random from \mathbf{F}_2 .

What is the fastest way to attack a uniform random system with large m and large n ? This question has an important application in post-quantum cryptography: solving such a system, in particular with m slightly smaller than n , conjecturally breaks a typical MQ signature system such as Patarin’s classic “HFE^{v-}” [19, Sect. 4 with modifications from 11.1 and 11.3]. Specifically, in these systems, the public key is a list of coefficients $\alpha_{i,j,k}$, conjectured to be difficult to distinguish from uniform random; the hash of a message is a list of coefficients β_i , which in the “random-oracle model” are uniform random by definition; and a solution to the corresponding system of equations is a signature on that message.

One of the central reasons for interest in these signature systems is that they allow very short signatures: it seems that a secure post-quantum MQ signature can be even shorter than a pre-quantum ECC signature. But the security evaluation here relies critically on quantifying the difficulty of solving a uniform random system of MQ equations.

There are also various proposals for MQ encryption systems where security analysis relies on a slight variant of the same question: m is taken somewhat larger than n , and one wants to know the fastest way to attack a uniform random *solvable* system. Algorithms designed to solve random systems of MQ equations have also had some applications beyond MQ cryptography, as illustrated by the attack in [8] against some small-key code-based encryption systems.

1.2 Performance of Various Algorithms for Random Systems. We consider asymptotic attack cost as $m \rightarrow \infty$ and $n \rightarrow \infty$ with an essentially constant ratio m/n . Specifically, let μ be a real number with $\mu \geq 1$, and assume that m is a function of n satisfying $m/n \in \mu + o(1)$ as $n \rightarrow \infty$.

All of the algorithmic issues that we analyze are visible for the frequently used case $\mu = 1$, and specifically $m = n$; the reader should feel free to focus on this case. Standard HFE^{v-} parameters actually take m slightly smaller than n but still have $m/n \in 1 + o(1)$ as $n \rightarrow \infty$. Beware, however, that “FXL” (see below) and “GroverXL” for $\mu = 1$ use “XL” for $\mu > 1$.

Brute-force search uses at most $N^{1+o(1)}$ operations where $N = 2^n$: there are N possibilities for (x_1, x_2, \dots, x_n) , and checking one possibility uses $N^{o(1)}$ operations. For $m < n$ (the “underdetermined” case) one can reasonably expect a solution to appear within just 2^m possibilities, but the assumption $\mu \geq 1$ means that 2^m does not beat $N^{1+o(1)}$.

Brute-force search is asymptotically beaten by Gröbner-basis techniques. In particular:

- “Extended linearization” (XL) uses just $N^{0.87280\dots+o(1)}$ operations in the case $\mu = 1$, under plausible assumptions that have been checked in various experiments.
- Even better, combining brute-force search with XL produces “fixing followed by extended linearization” (FXL), which uses just $N^{0.79106\dots+o(1)}$ operations in the case $\mu = 1$ under the same assumptions.

The exponents 0.87280... and 0.79106... here, modulo a calculation error (0.785 instead of 0.79106), were published by Yang, Chen, and Courtois in 2004 [26]. See Sect. 2 for further history and an explanation of how XL works.

Brute-force search is also asymptotically beaten by the recent Lokshtanov–Paturi–Tamaki–Williams–Yu algorithm [15], which uses at most $N^{0.8765+o(1)}$ operations. This algorithm is randomized but, for each input, is proven to produce the correct result with negligible chance of error. The exponent $0.8765+o(1)$ is above $0.79106... + o(1)$, and worst-case provability is outside the scope of our paper. We do not know whether the ideas in [15] can save time in FXL.

1.3 Quantum Algorithms for Random Systems. Quantum computers beat brute-force search in a different way: namely, Grover’s algorithm uses $N^{0.5+o(1)}$ operations. These operations are serial, but simply running A parallel copies of Grover’s algorithm reduces time by a factor $A^{1/2}$. For example, parallel Grover takes time $N^{0.46+o(1)}$ on a quantum computer of total area $N^{0.08+o(1)}$, or time $N^{0.35+o(1)}$ on a quantum computer of area $N^{0.3+o(1)}$.

The main question considered in this paper is whether Grover’s method can be usefully combined with XL. We answer this question in the affirmative. Our main contributions are the design and analysis of an algorithm “GroverXL” that, under the same assumptions used to analyze FXL, has exponent below $0.5+o(1)$.

We analyze this algorithm first in a simplified operation-count metric, and then in realistic area and time metrics for a parallel two-dimensional mesh-connected architecture. See Sects. 2.6 and 3.3 for discussion of the metrics, and Sect. 4 for the main analysis. For example, for $m = n$, GroverXL takes time $N^{t+o(1)}$ on a mesh-connected quantum computer of area $N^{a+o(1)}$, where the user can choose either of the following parameter sets (t, a) :

$$\begin{aligned} (t, a, t + a, t + a/2) &= (0.45742\dots, 0.01467\dots, 0.47210\dots, 0.46476\dots) \quad \text{or} \\ (t, a, t + a, t + a/2) &= (0.44962\dots, 0.02557\dots, 0.47519\dots, 0.46240\dots). \end{aligned}$$

The area-time product is $N^{t+a+o(1)}$, and parameter set 1 is designed to optimize this exponent $t + a$. GroverXL can be further parallelized, taking time $N^{t-p+o(1)}$ on a mesh-connected quantum computer of area $N^{a+2p+o(1)}$; parameter set 2 is designed to optimize this area-time tradeoff. For example, the time exponent drops to 0.35 with area exponent 0.22481..., whereas reaching time exponent 0.35 with parallel Grover needs area exponent 0.30000 as noted above.

We state our results more generally for systems of m quadratic equations in n variables over \mathbf{F}_q . The generalization from \mathbf{F}_2 to \mathbf{F}_q appears in many MQ systems and in further applications. Of course, guessing elements of \mathbf{F}_q becomes slower as q increases; for sufficiently large q , one should simply use XL.

2 XL and FXL

This section reviews the XL and FXL algorithms to solve m equations in n variables over a finite field \mathbf{F}_q . For simplicity we consider solely quadratic equations, although the ideas can easily be extended to cubic systems and higher.

$$\begin{array}{l}
 xyz + xy + xz + x = 0 \\
 0 = 0 \\
 xyz + xz + yz + z = 0 \\
 xy + x + yz + z = 0 \\
 xy + xz = 0 \\
 xyz + xy = 0 \\
 yz + z = 0 \\
 xz + x + y + 1 = 0 \\
 xyz + xy = 0 \\
 xyz + y = 0 \\
 xz + z = 0 \\
 xz + yz + y + z = 0
 \end{array}
 \begin{array}{c}
 \left[\begin{array}{c}
 11110000 \\
 00000000 \\
 10101010 \\
 01011010 \\
 01100000 \\
 11000000 \\
 00001010 \\
 00110101 \\
 11000000 \\
 10000100 \\
 00100010 \\
 00101110
 \end{array} \right]
 \begin{array}{c}
 [xyz] \\
 xy \\
 xz \\
 x \\
 yz \\
 y \\
 z \\
 1
 \end{array}
 = 0
 \end{array}
 \begin{array}{c}
 \left[\begin{array}{c}
 11110000 \\
 01011010 \\
 00111010 \\
 00010100 \\
 00001010 \\
 00000101 \\
 00000011 \\
 00000000 \\
 00000000 \\
 00000000 \\
 00000000 \\
 00000000
 \end{array} \right]
 \begin{array}{c}
 [xyz] \\
 xy \\
 xz \\
 x \\
 yz \\
 y \\
 z \\
 1
 \end{array}
 = 0
 \end{array}$$

Fig. 2.2. Small example of XL. The goal is to find a solution to the following system of three equations in three variables x, y, z over \mathbf{F}_2 : $xy + x + yz + z = 0$; $xz + x + y + 1 = 0$; $xz + yz + y + z = 0$. Left column (black): Twelve equations obtained as $x, y, z, 1$ times each of the original equations; note that x^2, y^2, z^2 are replaced with x, y, z respectively. Middle column (blue): Same twelve equations expressed in matrix form; the matrix is a Macaulay matrix. Right column (green): Equations obtained by applying Gaussian elimination to the Macaulay matrix. Three of the resulting equations are $x + y = 0$, $y + 1 = 0$, and $z + 1 = 0$, implying $(x, y, z) = (1, 1, 1)$. This is a solution, and therefore the only solution, to the original system. (Color figure online)

This section also analyzes the asymptotic performance of XL and FXL, assuming that m , the XL degree parameter d , and the FXL fixing parameter f grow linearly with n . In particular, this section reviews the asymptotic number of monomials and the asymptotic cost of linear algebra. See Sect. 3 for quantum speedups, and Sect. 4 for analysis of the overall costs for random systems when d/n and f/n are optimized.

2.1 XL: Extended Linearization. XL was introduced by Lazard [13]. It was rediscovered and given the name “XL” in [6].

XL begins by computing a **degree- d Macaulay matrix** as follows. Multiply each of the original m quadratic equations by each monomial of degree at most $d - 2$; each product is called a “relation”. Each relation is a linear combination of monomials of degree at most d . The Macaulay matrix is, by definition, the matrix of coefficients in these linear combinations. See Fig. 2.2 for a small example with $d = 3$.

If the relations have a linear combination of the form $1 = 0$ then the original system of equations has no solution. XL recognizes this situation by linear

algebra on the Macaulay matrix: it checks whether the vector $(0, 0, \dots, 1)$, with 1 at the position of monomial 1, is a linear combination of the rows of the matrix.

More generally, XL checks whether the relations have a nonzero linear combination involving only monomials of degree at most 1; i.e., whether the relations imply a nonzero linear equation among the variables. This linear equation reduces the original system to a smaller system that can be solved recursively (and further independent equations reduce the system even more). Recognizing this situation is again linear algebra on the Macaulay matrix.²

An alternative is to check whether the relations have a nonzero linear combination involving only powers of a single variable. The resulting univariate equation is easily solved by fast root-finding algorithms, and each root produces a smaller system that can be solved recursively. There can be “fake” roots that do not correspond to solutions of the original system, but experiments suggest that for random systems these “fake” roots rapidly produce contradictions in subsequent levels of recursion.

There is no guarantee that XL will produce any of this information. Increasing d can produce more information, but increasing d also produces many more monomials, as discussed below. A common way to use XL is to try $d = 2$, then $d = 3$, and so on, until the system is solved. As d increases, there appears to be a sharp transition from (1) XL solving very few systems to (2) XL solving almost all systems; the transition point is quantified in Sect. 4.

2.3 The Number of Monomials, and the Field Equation. A basic combinatorics exercise states that the number of monomials of degree $\leq d$ in n variables v_1, v_2, \dots, v_n is exactly the binomial coefficient $\binom{n+d}{d}$: the monomial $v_1^{e_1} v_2^{e_2} \dots v_n^{e_n}$ with $e_1 + e_2 + \dots + e_n \leq d$ corresponds to the d -element subset $\{e_1 + 1, (e_1 + 1) + (e_2 + 1), \dots, (e_1 + 1) + \dots + (e_n + 1)\}$ of $\{1, 2, \dots, n + d\}$.

If q is small then one can save time in XL and FXL by using the field equation $v^q = v$ to eliminate monomials with exponents larger than $q - 1$. For example, if $q = 2$ then one uses only squarefree monomials; if v^2 appears then one immediately replaces it with v . There are only $\binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{d}$ squarefree monomials of degree $\leq d$. The example in Fig. 2.2 uses this speedup.

More generally, define $\varphi_q \in \mathbf{Z}[z]$ as the polynomial $(1 - z^q)/(1 - z) = 1 + z + \dots + z^{q-1}$. The number of monomials of degree d in n variables with exponent at most $q - 1$ is the coefficient of z^d in φ_q^n , which we abbreviate $[z^d]\varphi_q^n$. The number of monomials of degree $\leq d$ is $\sum_{k \leq d} [z^k]\varphi_q^n$, or equivalently $[z^d]((1 + z + z^2 + \dots)\varphi_q^n)$.

² Part of the literature suggests, incorrectly, that this requires computing echelon form. In fact, it simply requires solving linear equations. Specifically, finding x such that Mx is zero outside $n + 1$ positions is the same as finding x such that $M'x = 0$, where M' removes those positions from M . To find a uniform random r such that $M'r = 0$, one can take a uniform random v , compute $M'v$, use any method to find a solution x to $M'x = M'v$, and compute $r = x - v$. Then $M'r$ is sampled uniformly at random from the space of vectors Mx that are zero outside the specified positions. If the space has positive dimension then each r has at least a 50% chance of discovering this.

The asymptotic behavior of the binomial coefficient $\binom{n}{d}$ is singly exponential in n when d is linear in n . The same is true more generally for $[z^d]\varphi_q^n$.

Specifically, assume that $d/n \in \delta + o(1)$ as $n \rightarrow \infty$, where $0 < \delta < q - 1$. Then the number of monomials of degree d in n variables with exponents at most $q - 1$ is $2^{(\text{mon}_q(\delta) + o(1))n}$, where mon_q is defined below. The number of monomials of degree $\leq d$ in n variables with exponents at most $q - 1$ is also $2^{(\text{mon}_q(\delta) + o(1))n}$ if $\delta \leq (q - 1)/2$, and $2^{(\lg q + o(1))n} = (q + o(1))^n$ for all $\delta \geq (q - 1)/2$, where \lg means \log_2 .

The definition of $\text{mon}_q(\delta)$ for $0 < \delta < q - 1$ is as follows: $\text{mon}_q(\delta) = \lg(\varphi_q(\rho)/\rho^\delta)$, where $\rho \in \mathbf{R}$ is the unique positive root of the polynomial

$$\left(\frac{z}{1-z} - \frac{qz^q}{1-z^q} - \delta \right) \varphi_q = -\delta + (1-\delta)z + (2-\delta)z^2 + \dots + (q-1-\delta)z^{q-1} \in \mathbf{R}[z].$$

To see that this polynomial has a positive root, observe that the constant coefficient $-\delta$ is negative while the top coefficient $q - 1 - \delta$ is positive. To see that the root is unique, observe that $z/(1-z) - qz^q/(1-z^q) - \delta$ is an increasing function of z when z is positive (its derivative is $1/(1-z)^2 + q^2z^{q-1}/(1-z^q)^2 > 0$) and that φ_q is positive when z is positive.

It is sometimes convenient to also define $\text{mon}_q(0) = 0$ and $\text{mon}_q(q - 1) = 0$. Then mon_q is a continuous function on the interval $[0, q - 1]$.

For example, mon_2 is exactly the binary entropy function: $\text{mon}_2(\delta) = -\delta \lg \delta - (1 - \delta) \lg(1 - \delta)$. As $q \rightarrow \infty$, the values $\text{mon}_q(\delta)$ converge up to what one might call $\text{mon}_\infty(\delta)$, namely $(1 + \delta) \text{mon}_2(\delta/(1 + \delta)) = (1 + \delta) \lg(1 + \delta) - \delta \lg \delta$.

As a more complicated example, $\text{mon}_3(\delta) = \lg(1 + \rho + \rho^2) - \delta \lg \rho$, where ρ is the unique positive root of the polynomial $-\delta + (1 - \delta)z + (2 - \delta)z^2$; i.e., $\rho = (\delta - 1 + \sqrt{1 + 6\delta - 3\delta^2})/(2(2 - \delta))$. If $d/n \in \delta + o(1)$ as $n \rightarrow \infty$ then the number of monomials of degree d in n variables with exponents at most 2 is $((1 + \rho + \rho^2)/\rho^\delta + o(1))^n$.

2.4 Understanding mon_q : The Saddle-Point Method. The fact that mon_q is the asymptotic exponent for the number of monomials follows from a standard trick in analytic combinatorics called the “saddle-point method”. For mon_q it is enough to apply a simple case of this method, making assumptions that we quote from [9, Sect. VIII.8.1]:

- B and C are power series with nonnegative coefficients. Our mon_q application takes $B = \varphi_q$ and $C = 1$.
- The constant coefficient of B is nonzero.
- The nonzero coefficients of B are at indices whose greatest common divisor is 1.
- B has a positive radius R of convergence in the complex plane. For us $R = \infty$.
- C has radius of convergence $\geq R$.
- T is the limit of $zB'(z)/B(z)$ as z approaches R from below. For us $T = q - 1$.

The saddle-point method then states the following. Fix δ with $0 < \delta < T$. If δn is an integer, then the coefficient $[z^{\delta n}]C(z)B(z)^n$ is $(c + o(1))C(\rho)B(\rho)^n/\rho^{\delta n+1}\sqrt{n}$

as $n \rightarrow \infty$, where ρ is the unique positive root of $\rho B'(\rho)/B(\rho) = \delta$, and c is an explicit nonzero constant. See [9, Proposition VIII.8].

In particular, for any δ strictly between 0 and $q - 1$, the coefficient $[z^{\delta n}] \varphi_q^n$ is $(c + o(1))/\rho\sqrt{n}$ times the n th power of $\varphi_q(\rho)/\rho^\delta$, where ρ is the unique positive root of $\rho\varphi'_q(\rho)/\varphi_q(\rho) = \delta$, i.e., $\rho/(1 - \rho) - q\rho^q/(1 - \rho^q) = \delta$.

This is called the “saddle-point method” as a reference to the name “saddle points” for roots of the derivative of an analytic function. The connection to saddle points arises as follows. Cauchy’s integration formula states that

$$[z^m]C(z)B(z)^n = \frac{1}{2\pi i} \oint \frac{C(z)B(z)^n dz}{z^{m+1}} = \frac{1}{2\pi i} \oint \frac{C(z)F(z)^n dz}{z},$$

where \oint integrates on any contour circling once (counterclockwise) around the origin in the complex plane, and where $F(z) = B(z)/z^{m/n}$. The saddle-point method chooses a contour that passes through one or more saddle points of $\log F(z)$, i.e., through roots of $B'(z)/B(z) - m/nz$: in the simple case mentioned above, this contour is a circle of radius ρ around the origin, passing through the unique positive root of $zB'(z)/B(z) = \delta$. One can show, under reasonable assumptions, that the integral is asymptotically dominated by the portion of the integral around the saddle points. For a full exposition see, e.g., [24] or [9, Chap. 8].

2.5 Fast Linear Algebra. Write A for the number of monomials analyzed above: the number of monomials of degree $\leq d$ in n variables with exponents at most $q - 1$. Each of the m equations provided as input to XL produces at most A relations, namely one relation for each monomial of degree at most $d - 2$. The total number of relations is at most mA . In other words, the Macaulay matrix has A columns and at most mA rows. We focus on the situation that A is exponential in n while m is linear in n ; the matrix then has $A^{1+o(1)}$ rows and columns.

Each of the original equations is assumed to be quadratic, and therefore has $O(n^2)$ terms. Consequently each relation also has $O(n^2)$ terms: i.e., there are only $O(n^2)$ nonzero entries in each row of the Macaulay matrix. The Macaulay matrix is thus extremely sparse.

Sparsity saves time in linear algebra. The fastest methods known to solve an $A \times A$ dense system of linear equations use $A^{\omega+o(1)}$ operations where $\omega \approx 2.37$, while sufficient sparsity reduces the number of operations to $A^{2+o(1)}$. The idea of applying sparse linear-algebra techniques to speed up XL was mentioned by Yang and Chen in 2004 [25], analyzed in more detail by Yang, Chen, and Courtois later the same year [26], and demonstrated in various XL implementations starting in 2006; see, e.g., [4].

We focus exclusively on Wiedemann’s algorithm [23] for sparse linear algebra over finite fields. The algorithm is described in [23, p. 59] as an “ $O(n_0(\omega + n_1 \log n_1) \log n_0)$ expected time method of producing a solution to any linear system [over \mathbf{F}_q], providing a solution exists”. Here “ ω ” is the total number of nonzero entries in the matrix; “ n_0 ” and “ n_1 ” are the minimum and maximum of the number of rows and the number of columns; and “time” counts operations

in a sequential RAM model, with each addition and multiplication in \mathbf{F}_q taking time 1.

In the XL situation mentioned above (m linear in n , and A exponential in n), “ n_0 ” and “ n_1 ” and “ ω ” are all bounded by $A^{1+o(1)}$, so the number of operations in Wiedemann’s algorithm is $A^{2+o(1)}$. A closer look shows that the algorithm is bottlenecked by a series of $A^{1+o(1)}$ matrix-vector multiplications, using vector length $A^{1+o(1)}$. The matrix and all other intermediate quantities also fit into $A^{1+o(1)}$ field elements.

2.6 Communication Costs and Parallelization. Each of the sparse matrix-vector multiplications described above consists of $A^{1+o(1)}$ random accesses to an array of $A^{1+o(1)}$ field elements.³ A simplified operation-count metric states that each random access has cost 1, independent of A .

This operation-count metric is a poor predictor of the time spent on computation, for two basic reasons. First, if one is spending $A^{1+o(1)}$ dollars on computer hardware, then one can afford as many as $A^{1+o(1)}$ small processing cores that operate in parallel; this could reduce the time by a factor as large as $A^{1+o(1)}$ if there is enough work to do in parallel. Second, the distance between array elements is forced to grow as a positive power of A , correspondingly increasing the time necessary for communication.

There are many previous papers designing algorithms for a two-dimensional $A^{0.5+o(1)} \times A^{0.5+o(1)}$ mesh of small parallel processing cores, with each core connected locally to its neighbors. For example, Brent and Kung showed in [3] how to multiply A -bit integers in time $A^{0.5+o(1)}$, and there are several papers showing how to sort A small items in time $A^{0.5+o(1)}$.

Sorting can in turn be used to implement a batch of random accesses, and in particular to parallelize Wiedemann’s algorithm in this model, as Bernstein [2] pointed out in the context of integer factorization. This reduces the time for Wiedemann’s algorithm, and the time for XL, to $A^{1.5+o(1)}$.

2.7 FXL: Fixing Followed by Extended Linearization. FXL was proposed by Courtois, Klimov, Patarin, and Shamir in 2000 [6].

FXL solves a system of m quadratic equations in n variables v_1, v_2, \dots, v_n over \mathbf{F}_q as follows. There are q^f possibilities for the last f variables v_{n-f+1}, \dots, v_n . For each possibility, use XL to solve the resulting system of m quadratic equations in $n - f$ variables. In other words, guess (fix) f variables before running XL.

Increasing f by 1 costs a factor q in the number of guesses. However, it also increases the ratio $m/(n-f)$, and this often has a benefit of decreasing the degree d needed for XL to succeed. Optimized FXL exponents for random systems are presented in Sect. 4.

Note that FXL can be trivially parallelized, reducing the time by any desired factor up through q^f at the expense of increasing area by a similar factor.

³ As q grows, one has to account for the growing cost of reading, writing, and arithmetic on field elements. For simplicity we focus on asymptotic statements as $n \rightarrow \infty$ with q fixed.

3 ReversibleXL and GroverXL

It is conceptually straightforward to replace the brute-force search in FXL with Grover’s quantum search method, reducing the number of search iterations to its square root. However, Grover’s method requires the underlying function—the function that is evaluated on an input to see whether the input is a solution to the search—to be computed *reversibly*, with no data ever erased.

The XL computation described in the previous section does not fit this model. The computation is constantly erasing data: it repeatedly overwrites a vector with a matrix-vector product. This section analyzes the costs of fitting XL into Grover’s method.

3.1 Reversible Computation. There would be no difficulty if our goal were merely to count operations: simply keep a journal of all intermediate results, and then run the computation again in reverse order, as in [1, Lemma 1].

Formally, any sequence of NANDs is converted into a reversible computation as follows. Say there are input bits b_1, b_2, \dots, b_i , followed by NANDs $b_{i+1} = 1 - b_{f(i+1)}b_{g(i+1)}$, $b_{i+2} = 1 - b_{f(i+2)}b_{g(i+2)}$, and so on through $b_T = 1 - b_{f(T)}b_{g(T)}$, where $f(j) < j$ and $g(j) < j$. Some of the bits are specified to be output bits; for simplicity assume that these are not the input bits and are not used for further computations inside this sequence of NANDs.

Consider the following reversible circuit applied to T bits b_1, b_2, \dots, b_T . First apply the following “NOT-Toffoli” gates: $b_{i+1} \leftarrow b_{i+1} + 1 - b_{f(i+1)}b_{g(i+1)}$; $b_{i+2} \leftarrow b_{i+2} + 1 - b_{f(i+2)}b_{g(i+2)}$; and so on through $b_T \leftarrow b_T + 1 - b_{f(T)}b_{g(T)}$. Then apply the same gates again in reverse order to the **ancilla** bits, i.e., the bits that are *not* output bits.

If bits b_{i+1}, \dots, b_T all start as 0 then the reversible circuit first computes exactly what the original NANDs did, and then it sets the ancilla bits back to 0. More generally, if the ancilla bits all start as 0 then the reversible circuit adds the output of the original function into the output bits, while leaving the input bits untouched and setting the ancilla bits back to 0.

In short, this reversible computation produces $(x, 0, F(x) + y)$ if the input is $(x, 0, y)$. This is what it means to compute a function F reversibly.

3.2 Saving Space: The Bennett–Tomba Conversion. The circuit described above uses T bits of storage (or T qubits in the context of Grover’s method). This amount of hardware is usually vastly larger than the amount of hardware needed for the original computation: in particular, the storage for XL expands quadratically. This begs the question of whether one can afford this amount of hardware, and the question of whether the same amount of hardware can be more productively used in other ways.

Bennett proved in [1, Theorem 1] that any computation using time T and space S can be converted into a reversible computation using time $O(T^{\log_2 3})$ and space $O(S \log T)$. Bennett also proved, with credit to Tompa, that $\log_2 3$ can be replaced by $1 + \epsilon$ for any $\epsilon > 0$. Bennett’s theorem is stated for multitape Turing machines; we return below to issues of parallelization and communication costs.

Bennett’s $\log_2 3$ conversion works as follows. Decompose a computation into two halves: specifically, say the output is $C_2(C_1(x))$, where x is the input. Starting from $(x, 0, y)$, reversibly compute C_1 by the same construction recursively, obtaining $(x, C_1(x), y)$; reversibly compute C_2 by the same construction recursively, obtaining $(x, C_1(x), C_2(C_1(x)) + y)$; and then reversibly compute C_1 again, obtaining $(x, 0, C_2(C_1(x)) + y)$ as desired. This takes three half-size computations. The required space is proportional to the number of levels of recursion; the point here is that ancillas used for C_1 are reused for C_2 .

More generally, the Bennett–Tomba conversion splits a computation into k parts, each taking time (approximately) T/k . Starting from $(x, 0, \dots, 0, y)$, compute C_1 , then C_2 , and so on through C_k , obtaining

$$(x, C_1(x), C_2(C_1(x)), \dots, C_{k-1}(\dots C_1(x) \dots), C_k(\dots C_1(x) \dots) + y).$$

Then compute C_{k-1} , then C_{k-2} , and so on through C_1 , obtaining

$$(x, 0, \dots, 0, C_k(\dots(x) \dots) + y).$$

This is the same strategy used above for the extreme case that each C_i is a single NAND. Allowing larger computations C_i produces time exponent $\log_k(2k - 1)$, while increasing the space by a factor that depends on k .

Taking $k \in 2^{\Theta(\sqrt{\log T})}$ produces time and space within factors $2^{O(\sqrt{\log T})}$ of the original computation, as pointed out by Knill in [12, Theorem 2.12]. In the context of our XL analyses, these factors are $2^{o(n)}$ and therefore do not affect our asymptotic exponents. Knill also pointed out some smaller optimizations that are not visible in our exponents.

3.3 Parallelizing the Bennett–Tomba Conversion. We point out that the idea of the Bennett–Tomba conversion is compatible with massively parallel computation and local communication, in particular communication on a realistic two-dimensional mesh architecture.

Assume that the original computation is a sequence of T time steps, where each step is carried out in parallel by A small processing cores. The cores are arranged in a $\sqrt{A} \times \sqrt{A}$ mesh, with edges between adjacent cores. Formally, core (i, j) has state $s[i, j, t]$ at time t consisting of a small number of bits; $s[i, j, t + 1]$ is the output of a small computation applied to $s[i, j, t]$, $s[i - 1, j, t]$, $s[i, j - 1, t]$, $s[i + 1, j, t]$, $s[i, j + 1, t]$, with states past the edge defined to be empty. For our XL application, “small” can be defined as subexponential in n , while T and A grow exponentially with n .

We convert this into a reversible computation on a $\sqrt{A} \times \sqrt{A}$ mesh of small cores as follows. Divide the original computation into k parts C_1, C_2, \dots, C_k , each taking time approximately T/k . Core (i, j) starts with $(s[i, j, 0], 0, \dots, y_{i,j})$. Recursively apply C_1 reversibly, recursively apply C_2 reversibly, and so on through C_k , obtaining

$$(s[i, j, 0], s[i, j, t_1], \dots, s[i, j, t_k] + y_{i,j}).$$

Then apply C_{k-1} reversibly, apply C_{k-2} reversibly, and so on through C_1 , obtaining

$$(s[i, j, 0], 0, \dots, s[i, j, t_k] + y_{i,j})$$

as desired. The base case of the recursion is a time-1 parallel computation consisting of small local computations, each of which is applied reversibly with small overhead.

This conversion visibly expands the state in each core. The final state is accompanied by a journal of k earlier states; and each level of recursion needs its own journal, overall multiplying the state size by $k(\log T)/\log k$. Each level of recursion also multiplies the time by $(2k-1)/k$. As in [12], we take $k \in 2^{\Theta(\sqrt{n})}$, so that the overall area and time overheads are subexponential in n . The expanded area also implies a slowdown in communication, but this is again subexponential in n .

3.4 ReversibleXL and GroverXL. ReversibleXL is, by definition, the result of applying the above parallel conversion to the XL computation of whether a system has a solution. As noted in Sect. 2, XL can fail to determine whether a system has a solution, but we assume that d is chosen large enough that XL works for all systems provided to ReversibleXL; see Sect. 4.

GroverXL solves a system of quadratic equations in n variables as follows. Use Grover’s method to search through the q^f possibilities for the last f variables, applying ReversibleXL to each possibility. If the system has a solution then Grover’s method returns a random choice of solution; otherwise it returns a uniform random choice of the last f variables. Either way, substitute this choice into the system, and use XL to see whether there is a solution for the remaining variables.

GroverXL takes the time for $q^{f/2+o(1)}$ quantum computations of ReversibleXL, plus a final computation of XL. Like other applications of Grover’s method, GroverXL can be parallelized across many separate computations, increasing the area by a corresponding factor while dividing the time by the square root of the same factor. The limit of “many” is q^f , at which point one should simply use FXL.

4 Analysis for Random Systems

This section presents asymptotic cost exponents for solving random systems of $(\mu + o(1))n$ quadratic equations in n variables over \mathbf{F}_q , assuming $\mu \geq 1$. Exponents are shown for various small choices of q and for various choices of μ ranging from 1.0 up through 2.0.

Exponent e means that the cost is $2^{(e+o(1))n}$ as $n \rightarrow \infty$, or equivalently $(2^e + o(1))^n$. Simple brute-force search has exponent $\lg q$, and Grover’s algorithm has exponent $0.5 \lg q$, where as before $\lg = \log_2$. In all cases GroverXL has better exponents.

4.1 A Script for Computing Cost Exponents. To simplify verification, and to let the reader easily compute cost exponents for further pairs (q, μ) , we include a script to compute exponents. See Figs. 4.2 and 4.3. This script uses the free Sage computer-algebra system, version 8.0.

The script covers both GroverXL and FXL. In each case it covers two different metrics: (1) the exponent of a simplified operation count, and (2) the exponent

```

import collections

# generic caching mechanism
class memoized(object):
    def __init__(self,func):
        self.func = func
        self.cache = {}
        self.__name__ = 'memoized:' + func.__name__
    def __call__(self,*args):
        if not isinstance(args,collections.Hashable):
            return self.func(*args)
        if not args in self.cache:
            self.cache[args] = self.func(*args)
        return self.cache[args]

@memoized
def lg(x):
    return log(x*1.0)/log(2.0)

Zx.<x> = ZZ[]
Ry.<y> = RR[]
Zxz.<z> = Zx[]
Zxxa.<a> = Zxz[]

@memoized
def phi(q): # see Section 2.3
    return Zx((1-x^q)/(1-x))

@memoized
def monpoly(q): # see Section 2.3
    h = x/(1-x) - q*x^q/(1-x^q)
    return Zx(phi(q)*h)

@memoized
def mon(q,delta): # see Section 2.3
    if not q in ZZ: raise Exception('q must be integer')
    if q < 2: raise Exception('q must be at least 2')
    if delta < 0: return -Infinity
    if delta == 0: return 0
    if delta == q-1: return 0
    if delta > q-1: return -Infinity
    g = Ry(monpoly(q)) - delta*Ry(phi(q))
    roots = g.roots(RR)
    rho = max(r for r,e in roots)
    return lg(phi(q)(rho)/rho^delta)

```

Fig. 4.2. Script for computing cost exponents, part 1: caching and mon computation.

of the area-time product AT on a two-dimensional mesh-connected computer. In the context of parallelizing Grover's method, another metric of interest is the exponent of the \sqrt{AT} product; but for parallelized GroverXL this turns out to be identical to the first metric.

The script tries five values of q , namely 2, 3, 4, 5, 16; this is specified by `doit(2)` through `doit(16)` at the end of the script. The script takes a few hours to run, almost entirely for $q = 16$.

There are three nested loops for each q : `search` is either 0.5 for GroverXL or 1 for FXL; `linalg` is either 2 for a simplified operation-count metric or 2.5 for area-time product on a two-dimensional mesh; and `k` tries each μ between 1 and 2 in steps of 0.01. For each choice of $(q, \text{search}, \text{linalg}, \mu)$, the script prints

```

@memoized
def deltapoly(q): # see Section 4.5
    h = (-x/z - q*z^(q-1)/(1-z^q) + 1/(1-z)
        - 2*a*z/(1-z^2) + 2*a*q*z^(2*q-1)/(1-z^(2*q)))
    h *= z*(1-z^(2*q))/(1-z)
    return Zxza(h)

@memoized
def delta(q,mu): # see Section 4.5
    hmu = deltapoly(q)(QQ(mu)).discriminant()
    roots = hmu.roots(RR)
    if not roots: return -1
    return min(r for r,e in roots if r>0)

@memoized
def alpha(q,mu): # see Section 4.4
    return mon(q,delta(q,mu))

def doit(q):
    for search in [0.5,1]:
        searchlgq = search * lg(q)
        for linalg in [2,2.5]:
            def f(x): return (linalg*alpha(q,x) - searchlgq)/x
            bestvalue,mu0 = find_local_minimum(f,1,10)
            mu0 = RR(mu0)
            for k in range(100,201):
                mu = k*0.01
                x = max(mu,mu0) # lambda in Section 4.6
                context = '%d %.1f %.1f' % (q,search,linalg)
                cost = mu*f(x) + searchlgq
                alphax = alpha(q,x)
                print context, '%.3f' % mu, cost, x, alphax, alphax*mu/x
                sys.stdout.flush()

doit(2)
doit(3)
doit(4)
doit(5)
doit(16)

```

Fig. 4.3. Script for computing cost exponents, part 2: δ optimization and μ_0 optimization.

one line showing the exponent for solving $m = (\mu + o(1))n$ random quadratic equations in n variables over \mathbf{F}_q .

4.4 Understanding the XL Exponent. Guessing variables does not save time if the system is sufficiently overdetermined: i.e., if μ is larger than a particular cutoff μ_0 then FXL and GroverXL both boil down to XL. The script computes the cost exponent for XL in three steps:

- Compute δ as explained in Sect. 4.5. The XL degree d is $(\delta + o(1))n$.
- Compute $\alpha = \text{mon}_q(\delta)$. The number of monomials in XL is $2^{(\alpha+o(1))n}$.
- The exponent is $\text{linalg} \cdot \alpha$.

Specifically, XL takes time $T = 2^{(1.5\alpha+o(1))n}$ on a mesh of area $A = 2^{(\alpha+o(1))n}$, so AT is $2^{(2.5\alpha+o(1))n}$. In a simplified operation-count metric the exponent is only 2α . Note that the \sqrt{AT} exponent is also 2α ; as mentioned above, the \sqrt{AT} exponent is identical to the simplified operation-count exponent for these algorithms.

4.5 Understanding δ . The script uses XL degree $d \in (\delta + o(1))n$, where δ is computed as follows. Define h as the polynomial

$$z \frac{1 - z^{2q}}{1 - z} \left(\frac{-x}{z} - \frac{qz^{q-1}}{1 - z^q} + \frac{1}{1 - z} - \frac{2\mu z}{1 - z^2} + \frac{2\mu q z^{2q-1}}{1 - z^{2q}} \right)$$

in the polynomial ring $\mathbf{R}[x, z]$. Define $\Delta \in \mathbf{R}[x]$ as the discriminant of h with respect to z . Then Δ has a unique positive real root, namely δ .

This is a concise statement of a calculation explained in the previous XL literature. For example, for $q = 2$ and $m = n$, the XL exponent 0.87280... and the FXL exponent 0.79106... were calculated this way in [26]. The rest of this subsection reviews the main steps in the argument that this is the correct asymptotic degree for XL.

Recall that A , the number of monomials in XL, is the coefficient of z^d in $\varphi_q(z)^n/(1 - z)$, where $\varphi_q(z) = (1 - z^q)/(1 - z)$; in short, $[z^d](\varphi_q(z)^n/(1 - z))$. The number of relations is at most $m[z^{d-2}](\varphi_q(z)^n/(1 - z))$. A more careful analysis shows that the linear span of the relations has codimension (i.e., A minus the dimension) at least $[z^d](\varphi_q(z)^n/(1 - z)\varphi_q(z^2)^m)$. See [25, Theorem 2]; see also [7].

As d increases, there is a sharp transition in the behavior of the coefficient $[z^d](\varphi_q(z)^n/(1 - z)\varphi_q(z^2)^m)$, and in the experimentally observed behavior of XL for random systems. If d is noticeably below a cutoff analyzed below, then the coefficient is a huge positive integer, and XL almost always fails for random systems (although it does succeed for some special systems of interest): there are not enough relations to provide interesting information about any small subset of the monomials. As d grows past the cutoff, the coefficient crosses below 0 and rapidly becomes quite negative, and XL almost always succeeds for random systems. If the coefficient happens to be extremely close to 0 then XL will often succeed and often fail (there are often some accidental extra dependencies between relations), but adding $o(n)$ to d eliminates this vacillation.

We analyze the asymptotics of this coefficient as in Sect. 2.4. First use Cauchy’s integration formula

$$[z^d] \left(\frac{\varphi_q(z)^n}{(1 - z)\varphi_q(z^2)^m} \right) = \frac{1}{2\pi i} \oint \frac{\varphi_q(z)^n dz}{z^{d+1}(1 - z)\varphi_q(z^2)^m} = \frac{1}{2\pi i} \oint \frac{F(z)^n dz}{z(1 - z)}$$

where $F(z) = \varphi_q(z)/z^{d/n}\varphi_q(z^2)^{m/n}$. Then substitute $d = \delta n$ and $m = \mu n$, and apply the saddle-point method to compute an asymptotic formula for the integral as $n \rightarrow \infty$. This asymptotic formula involves powers of the form $F(\rho)^n$, where ρ runs through the complex roots of the logarithmic derivative

$$\frac{F'(z)}{F(z)} = \frac{-\delta}{z} - \frac{qz^{q-1}}{1 - z^q} + \frac{1}{1 - z} - \frac{2\mu z}{1 - z^2} + \frac{2\mu q z^{2q-1}}{1 - z^{2q}}.$$

Multiplying this logarithmic derivative by $z(1 - z^{2q})/(1 - z)$ produces the polynomial h defined earlier, with δ substituted for x . The roots of h are essentially the roots of F'/F ; a closer look shows that h has an extra root -1 if q is odd, but this does not affect the calculation of the cutoff.

Finally, with some work one can see that the phase transition from positive coefficients to negative coefficients occurs exactly when h has a double root, i.e., exactly when the discriminant Δ is zero. See generally [5] for the theory of double saddle points, and [26] for applications to XL.

4.6 Understanding the FXL and GroverXL Exponents. More generally, for any $\mu \geq 1$, the script computes the cost exponents for FXL and GroverXL as follows:

- Choose $\lambda \geq \mu$ as explained below. Assume that $(1 - \mu/\lambda + o(1))n$ variables are fixed, i.e., that XL is given $(\mu + o(1))n$ equations in $(\mu/\lambda + o(1))n$ variables.
- Compute α as in Sect. 4.4, starting from λ rather than μ . Then XL and ReversibleXL take time $T = 2^{(1.5\alpha + o(1))(\mu/\lambda + o(1))n}$ using area $T = 2^{(\alpha + o(1))(\mu/\lambda + o(1))n}$; i.e., in base 2^n , they have time exponent $1.5\alpha\mu/\lambda$ and area exponent $\alpha\mu/\lambda$ (and operation-count exponent $2\alpha\mu/\lambda$).
- For FXL, add $(1 - \mu/\lambda) \lg q$ to the time exponent (and the operation-count exponent) to account for the cost of brute-force search. For GroverXL, add $0.5(1 - \mu/\lambda) \lg q$. In other words, add `search` $(1 - \mu/\lambda) \lg q$.

To summarize, the exponent is `linalg` · $\alpha\mu/\lambda + \text{search}(1 - \mu/\lambda) \lg q$, where α is implicitly a function of λ . This formula shows that λ is best chosen to minimize `linalg` · $\alpha/\lambda - \text{search}(\lg q)/\lambda$. The script uses Sage’s `find_local_minimum` to find the minimum of this function on $[1, 10]$; larger inputs did not help for the range of `linalg` etc. that we use. The position of this maximum is `mu0`, exactly the cutoff μ_0 mentioned above. The script then defines $\lambda = \max\{\mu, \mu_0\}$.

4.7 Example: GroverXL for $q = 2$. The polynomial h defined earlier is $(1 - 2\mu - \delta)z^3 + (-2\mu - \delta)z^2 + (1 - \delta)z - \delta$. The discriminant Δ of h with respect to z is a quartic polynomial in δ , so the equation $\Delta = 0$ can be solved explicitly by radicals, and it is easy to see the unique positive root:

$$\delta = F(\mu) = -\mu + \frac{1}{2} + \frac{1}{2} \sqrt{2\mu^2 - 10\mu - 1 + 2\sqrt{\mu^4 + 6\mu^3 + 12\mu^2 + 8\mu}} \quad (1)$$

For example, if $(q, \mu) = (2, 1)$, then $\delta = 0.0899798\dots$; and $\alpha = \text{mon}_2(\delta) = -\delta \lg \delta - (1 - \delta) \lg(1 - \delta) = 0.436402\dots$. This means that XL uses degree $2^{(0.08997\dots + o(1))n}$, and $2^{(0.43640\dots + o(1))n}$ monomials. The operation-count exponent is $2 \cdot 0.43640\dots = 0.87280\dots$.

As another example, $\mu_0 = 1.81626\dots$ maximizes $(1 - 2 \text{mon}_2(F(\mu_0)))/\mu_0$. For $\mu = \mu_0$, XL has $\delta = F(\mu_0) = 0.05573\dots$ and $\alpha = \text{mon}_2(\delta) = 0.31026\dots$, for operation-count exponent $0.62052\dots$.

FXL, when optimized for operation count, fixes enough variables to reach m/μ_0 remaining variables. For example, again for $(q, \mu) = (2, 1)$, FXL runs XL with $m = (1 + o(1))n$ equations and $(1/\mu_0 + o(1))n = (0.55058\dots + o(1))n$ variables. XL’s operation-count exponent in base 2^n is then $0.55058\dots \cdot 0.62052\dots = 0.34164\dots$. The remaining $(1 - 1/\mu_0 + o(1))n = (0.44941\dots + o(1))n$ variables are found by brute-force search, so the final exponent for FXL is $0.79106\dots$.

We emphasize that this calculation so far is not new: FXL was analyzed this way in [26]. Our main contributions are the design and analysis of GroverXL.

Table 4.10. GroverXL operation-count exponent for $q \in \{2, 3, 4, 5, 16\}$ and various μ . Operation count ignores communication costs. Each exponent is rounded down to multiple of 0.00001. For comparison, Grover’s algorithm without XL has exponents 0.50000, 0.79248, 1.00000, 1.16096, 2.00000, independently of μ .

	$\mu = 1.0$	$\mu = 1.1$	$\mu = 1.2$	$\mu = 1.3$	$\mu = 1.4$	$\mu = 1.5$	$\mu = 1.6$	$\mu = 1.7$	$\mu = 2.0$
$q = 2$	0.46240	0.45864	0.45488	0.45112	0.44737	0.44361	0.43985	0.43609	0.42481
$q = 3$	0.70425	0.69542	0.68660	0.67778	0.66895	0.66013	0.65131	0.64248	0.61601
$q = 4$	0.85848	0.84433	0.83018	0.81602	0.80187	0.78772	0.77357	0.75942	0.71696
$q = 5$	0.96843	0.94918	0.92993	0.91068	0.89142	0.87217	0.85292	0.83367	0.77591
$q = 16$	1.42604	1.36865	1.31125	1.25386	1.19646	1.13907	1.08167	1.02428	0.86575

Table 4.11. GroverXL cost exponent for $q \in \{2, 3, 4, 5, 16\}$ and various μ . Cost is area-time product on two-dimensional mesh-connected computer. Each exponent is rounded down to multiple of 0.00001. For comparison, Grover’s algorithm without XL has exponents 0.50000, 0.79248, 1.00000, 1.16096, 2.00000, independently of μ .

	$\mu = 1.0$	$\mu = 1.1$	$\mu = 1.2$	$\mu = 1.3$	$\mu = 1.4$	$\mu = 1.5$	$\mu = 1.6$	$\mu = 1.7$	$\mu = 2.0$
$q = 2$	0.47210	0.46931	0.46652	0.46373	0.46094	0.45815	0.45536	0.45257	0.44420
$q = 3$	0.72468	0.71790	0.71112	0.70434	0.69756	0.69078	0.68400	0.67722	0.65688
$q = 4$	0.88987	0.87886	0.86785	0.85683	0.84582	0.83481	0.82380	0.81278	0.77975
$q = 5$	1.01016	0.99508	0.98000	0.96492	0.94984	0.93476	0.91968	0.90460	0.85937
$q = 16$	1.53753	1.49128	1.44503	1.39879	1.35254	1.30629	1.26005	1.21380	1.07506

Our script is also new (and possibly the first public software to automate these analyses), as are the area-time analyses.

For GroverXL, we find minimum exponent 0.47210... for the area-time product by taking $\mu_0 = 7.74234...$ to maximize $(0.5 - 2.5 \text{mon}_2(F(\mu_0)))/\mu_0$. We also find minimum operation-count exponent 0.46240... by taking $\mu_0 = 5.63489...$ to maximize $(0.5 - 2 \text{mon}_2(F(\mu_0)))/\mu_0$.

4.8 Example: GroverXL for $q = 3$. The polynomial h is now $(-\delta + 2 - 4\mu)z^4 - z^3 + (-\delta + 1 - 2\mu)z^2 + z - \delta = 0$. The discriminant Δ is a degree-6 polynomial, and the equation $\Delta = 0$ is again solvable in radicals for δ as a function of μ . This solution is quite complex, presumably less efficient than the more general root-finding techniques used by our script.

Numerical computations proceed as in Sect. 4.7. For example, for $(q, \mu) = (3, 1)$ we find minimum area-time exponent 0.72468... (compared to 1.27507... for FXL) by taking $\mu_0 = 5.36509...$, and minimum operation-count exponent 0.70425... (compared to 1.17521... for FXL) by taking $\mu_0 = 4.11429...$

4.9 Tables of Results. Tables 4.10 and 4.11 show the GroverXL operation-count exponent and cost exponent respectively, as computed by the script from Sect. 4.1. Tables 4.12 and 4.13 show the exponents for the amount of hardware.

Table 4.12. GroverXL space exponent when parameters are optimized for operation count, for $q \in \{2, 3, 4, 5, 16\}$ and various μ . Operation count ignores communication costs. Each exponent is rounded down to multiple of 0.00001.

	$\mu = 1.0$	$\mu = 1.1$	$\mu = 1.2$	$\mu = 1.3$	$\mu = 1.4$	$\mu = 1.5$	$\mu = 1.6$	$\mu = 1.7$	$\mu = 2.0$
$q = 2$	0.02557	0.02812	0.03068	0.03324	0.03579	0.03835	0.04091	0.04346	0.05114
$q = 3$	0.05219	0.05741	0.06263	0.06785	0.07306	0.07828	0.08350	0.08872	0.10438
$q = 4$	0.07882	0.08670	0.09458	0.10247	0.11035	0.11823	0.12611	0.13400	0.15764
$q = 5$	0.10377	0.11415	0.12453	0.13490	0.14528	0.15566	0.16604	0.17641	0.20755
$q = 16$	0.26759	0.29434	0.32110	0.34786	0.37462	0.40138	0.42814	0.45490	0.43287

Table 4.13. GroverXL area exponent when parameters are optimized for cost, for $q \in \{2, 3, 4, 5, 16\}$ and various μ . Cost is area-time product on two-dimensional mesh-connected computer. Each exponent is rounded down to multiple of 0.00001.

	$\mu = 1.0$	$\mu = 1.1$	$\mu = 1.2$	$\mu = 1.3$	$\mu = 1.4$	$\mu = 1.5$	$\mu = 1.6$	$\mu = 1.7$	$\mu = 2.0$
$q = 2$	0.01467	0.01614	0.01760	0.01907	0.02054	0.02200	0.02347	0.02494	0.02934
$q = 3$	0.03196	0.03516	0.03835	0.04155	0.04475	0.04794	0.05114	0.05434	0.06393
$q = 4$	0.04992	0.05491	0.05990	0.06489	0.06988	0.07488	0.07987	0.08486	0.09984
$q = 5$	0.06696	0.07365	0.08035	0.08704	0.09374	0.10044	0.10713	0.11383	0.13392
$q = 16$	0.18512	0.20363	0.22215	0.24066	0.25917	0.27768	0.29620	0.31471	0.37025

Table 4.14. FXL operation-count exponent for $q \in \{2, 3, 4, 5, 16\}$ and various μ . Operation count ignores communication costs. Each exponent is rounded down to multiple of 0.00001. For comparison, brute-force search without XL has exponents 1.00000, 1.58496, 2.00000, 2.32192, 4.00000, independently of μ .

	$\mu = 1.0$	$\mu = 1.1$	$\mu = 1.2$	$\mu = 1.3$	$\mu = 1.4$	$\mu = 1.5$	$\mu = 1.6$	$\mu = 1.7$	$\mu = 2.0$
$q = 2$	0.79106	0.77017	0.74928	0.72838	0.70749	0.68660	0.66570	0.64481	0.58466
$q = 3$	1.17521	1.13423	1.09325	1.05228	1.01130	0.97033	0.92935	0.88839	0.78134
$q = 4$	1.39851	1.33836	1.27821	1.21807	1.15792	1.09777	1.03763	0.98102	0.84342
$q = 5$	1.54347	1.46563	1.38778	1.30993	1.23209	1.15424	1.07963	1.01421	0.86056
$q = 16$	2.00814	1.80896	1.60977	1.42959	1.29463	1.18842	1.10168	1.02899	0.86575

For example, the top-left entries in these tables are for $q = 2$ and $\mu = 1.0$. The entries are, respectively, 0.46240, 0.47210, 0.02557, and 0.01467. The first and third numbers indicate that GroverXL uses $2^{(0.46240 \dots + o(1))n}$ operations in space $2^{(0.02557 \dots + o(1))n}$, when GroverXL parameters are optimized for operation count. The second and fourth numbers indicate that GroverXL has area-time product $2^{(0.47210 \dots + o(1))n}$ using area $2^{(0.01467 \dots + o(1))n}$, when GroverXL parameters are optimized for area-time product.

For comparison, Tables 4.14 and 4.15 show the FXL operation-count exponent and cost exponent. Note that the case $q = 16$ and $\mu = 2.0$ has the same

Table 4.15. FXL cost exponent for $q \in \{2, 3, 4, 5, 16\}$ and various μ . Cost is area-time product on two-dimensional mesh-connected computer. Each exponent is rounded down to multiple of 0.00001. For comparison, brute-force search without XL has exponents 1.00000, 1.58496, 2.00000, 2.32192, 4.00000, independently of μ .

	$\mu = 1.0$	$\mu = 1.1$	$\mu = 1.2$	$\mu = 1.3$	$\mu = 1.4$	$\mu = 1.5$	$\mu = 1.6$	$\mu = 1.7$	$\mu = 2.0$
$q = 2$	0.85284	0.83813	0.82341	0.80870	0.79398	0.77926	0.76455	0.74983	0.70569
$q = 3$	1.27507	1.24409	1.21310	1.18211	1.15112	1.12013	1.08914	1.05816	0.96519
$q = 4$	1.52698	1.47968	1.43237	1.38507	1.33777	1.29047	1.24317	1.19587	1.05396
$q = 5$	1.69553	1.63289	1.57025	1.50761	1.44497	1.38233	1.31969	1.25705	1.07570
$q = 16$	2.29409	2.12350	1.95291	1.78232	1.61828	1.48552	1.37710	1.28624	1.08219

operation-count exponent for FXL as for GroverXL; in this case μ is above $\mu_0 \approx 1.80$, and guessing is not helpful (although it does help in area-time product, since then $\mu_0 \approx 2.16$). For smaller values of q and μ , GroverXL has better exponents than FXL, which in turn has better exponents than XL.

References

1. Bennett, C.H.: Time/space trade-offs for reversible computation. *SIAM J. Comput.* **18**, 766–776 (1989). Cited in §3.1, §3.2
2. Bernstein, D.J.: Circuits for integer factorization: a proposal (2001). https://cr.ypt.org/papers.html#nfs_circuit. Cited in §2.6
3. Brent, R.P., Kung, H.T.: The area-time complexity of binary multiplication. *J. ACM* **28**, 521–534 (1981). <http://www.maths.anu.edu.au/~brent/pub/pub055.html>. Cited in §2.6
4. Cheng, C.-M., Chou, T., Niederhagen, R., Yang, B.-Y.: Solving quadratic equations with XL on parallel architectures. In: CHES 2012 [21], pp. 356–373 (2012). <https://eprint.iacr.org/2016/412>. Cited in §2.5
5. Chester, C.R., Friedman, B., Ursell, F.: An extension of the method of steepest descents. *Proc. Camb. Philos. Soc.* **53**, 599–611 (1957). Cited in §4.5
6. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Eurocrypt 2000 [20], pp. 392–407 (2000). <http://minrank.org/xlfull.pdf>. Cited in §2.1, §2.7
7. Diem, C.: The XL-algorithm and a conjecture from commutative algebra. In: Asiacrypt 2004 [14], pp. 323–337 (2004). Cited §4.5
8. Faugère, J.-C., Otmani, A., Perret, L., Tillich, J.-P.: Algebraic cryptanalysis of McEliece variants with compact keys. In: Eurocrypt 2010 [10], pp. 279–298 (2010). <https://www.iacr.org/archive/eurocrypt2010/66320290/66320290.pdf>. Cited in §1.1
9. Flajolet, P., Sedgewick, R.: *Analytic Combinatorics*. Cambridge University Press, Cambridge (2009). ISBN 978-0-521-89806-5. <http://ac.cs.princeton.edu/home/>. Cited in §2.4, §2.4, §2.4
10. Gilbert, H. (ed.): *Advances in Cryptology-EUROCRYPT 2010*. LNCS, vol. 6110. Springer, Heidelberg (2010). <https://doi.org/10.1007/978-3-642-13190-5>. ISBN 978-3-642-13189-9. See [8]

11. Klein, P.N. (ed.): Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, 16–19 January. SIAM (2017). See [15]
12. Knill, E.: An analysis of Bennett’s pebble game (1995). <http://arxiv.org/abs/math/9508218>. Cited in §3.2, §3.3
13. Lazard, D.: Résolution des systèmes d’équations algébriques. *Theoret. Comput. Sci.* **15**, 77–110 (1981). <https://www.sciencedirect.com/science/article/pii/0304397581900645>. Cited in §2.1
14. Lee, P.J. (ed.): Advances in Cryptology-ASIACRYPT 2004. LNCS, vol. 3329. Springer, Heidelberg (2004). <https://doi.org/10.1007/b104116>. See [7]
15. Lokshtanov, D., Paturi, R., Tamaki, S., Williams, R.R., Yu, H.: Beating brute force for systems of polynomial equations over finite fields. In: SODA 2017, [11], pp. 2190–2202 (2017). <http://theory.stanford.edu/~yuhch123/files/polyEq.pdf>. Cited in §1.2, §1.2
16. Lopez, J., Qing, S., Okamoto, E. (eds.): Information and Communications Security, ICICS 2004. LNCS, vol. 3269. Springer, Cham (2004). <https://doi.org/10.1007/b101042>. ISBN 3-540-23563-9. See [26]
17. Maurer, U.M. (ed.): Advances in Cryptology-EUROCRYPT 1996. LNCS, vol. 1070. Springer, Heidelberg (1996). <https://doi.org/10.1007/3-540-68339-9>. ISBN 3-540-61186-X. MR 97g:94002. See [18]
18. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In: Eurocrypt 1996 [17], pp. 33–48 (1996). See also newer version [19]
19. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms, extended version (1998). See also older version [18]. <http://minrank.org/hfe.pdf>. Cited in §1.1
20. Preneel, B. (ed.): Advances in Cryptology-EUROCRYPT 2000. LNCS, vol. 1807. Springer, Heidelberg (2000). <https://doi.org/10.1007/3-540-45539-6>. See [6]
21. Prouff, E., Schaubert, P. (eds.): Cryptographic Hardware and Embedded Systems-CHES 2012. LNCS, vol. 7428. Springer, Heidelberg (2012). ISBN 978-3-642-33026-1. See [4]
22. Wang, H., Pieprzyk, J., Varadharajan, V. (eds.): Information Security and Privacy. LNCS, vol. 3108. Springer, Heidelberg (2004). <https://doi.org/10.1007/b98755>. ISBN 3-540-22379-7. See [25]
23. Wiedemann, D.H.: Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory* **32**, 54–62 (1986). MR 87g:11166. Cited in §2.5, §2.5
24. Wong, R.: Asymptotic Approximations of Integrals. Academic Press, Cambridge (1989). ISBN 0-12-762535-6. Cited in §2.4
25. Yang, B.-Y., Chen, J.-M.: Theoretical analysis of XL over small fields. In: ACISP 2004 [22], pp. 277–288 (2004). <http://precision.moscito.org/by-publ/recent/xxl2-update.pdf>. Cited in §2.5, §4.5
26. Yang, B.-Y., Chen, J.-M., Courtois, N.: On asymptotic security estimates in XL and Gröbner bases-related algebraic cryptanalysis. In: ICICS 2004, [16], pp. 401–413 (2004). <http://www.iis.sinica.edu.tw/papers/byyang/2384-F.pdf>. Cited in §1.2, §2.5, §4.5, §4.5, §4.7



Improved Quantum Information Set Decoding

Elena Kirshanova()

Laboratoire LIP, ENS de Lyon, Lyon, France
elena.kirshanova@ens-lyon.fr

Abstract. In this paper we present quantum information set decoding (ISD) algorithms for binary linear codes. First, we refine the analysis of the quantum walk based algorithms proposed by Kachigar and Tillich (PQCrypto'17). This refinement allows us to improve the running time of quantum decoding in the leading order term: for an n -dimensional binary linear code the complexity of May-Meurer-Thomae ISD algorithm (Asiacrypt'11) drops down from $2^{0.05904n+o(n)}$ to $2^{0.05806n+o(n)}$. Similar improvement is achieved for our quantum version of Becker-Jeux-May-Meurer (Eurocrypt'12) decoding algorithm. Second, we translate May-Ozerov Near Neighbour technique (Eurocrypt'15) to an 'update-and-query' language more common in a similarity search literature. This re-interpretation allows us to combine Near Neighbour search with the quantum walk framework and use both techniques to improve a quantum version of Dumer's ISD algorithm: the running time goes down from $2^{0.059962n+o(n)}$ to $2^{0.059450+o(n)}$.

Keywords: Information set decoding · Quantum walk
Near Neighbour

1 Introduction

The *Information Set Decoding problem* with integer parameters n, k, d asks to find the error-vector $\mathbf{e} \in \mathbb{F}_2^n$ given a matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ and a vector $\mathbf{s} = \mathbf{H}\mathbf{e}^t$ such that the Hamming weight of \mathbf{e} , denoted $w := wt(\mathbf{e})$, is bounded by some integer. The matrix \mathbf{H} is called the parity-check matrix of a binary linear $[n, k, d]$ -code \mathcal{C} , where d is the minimum distance of the code. In this work, we stick to the so-called *full distance decoding* setting, i.e., when we search for \mathbf{e} with $wt(\mathbf{e}) \leq d$. The analysis is easy to adapt to *half-distance decoding*, i.e., when $wt(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$.

The ISD problem is relevant not only in coding theory but also in cryptography: several cryptographic constructions, e.g. [McE78], rely on the hardness of ISD. The problem seems to be intractable even for quantum computers, which makes these constructions attractive for post-quantum cryptography.

First classical ISD algorithm due to Prange dates back to 1962 [Pra62] followed by a series of improvements [Ste89, Dum91, MMT11, BJMM12], culminating in algorithms [MO15, BM17] that rely on Nearest Neighbour techniques in

Hamming metric. On the quantum side, the ISD problem received much less attention: Bernstein in [Ber10] analysed a quantum version of Prange’s algorithm, and recently Kachigar and Tillich [KT17] gave a series of ISD algorithms based on quantum walks. The results presented here extend and improve upon the work of [KT17].

Our contributions:

1. We describe quantum versions of all known ISD algorithms, which have better asymptotical running times than algorithms given in [KT17]. We also explain why for certain algorithms the analysis of [KT17] is sub-optimal;
2. We re-phrase May-Ozerov Near Neighbour algorithm [MO15] in the ‘update-and-query’ language and give a method to analyse its complexity;
3. We present a quantum version of the May-Ozerov ISD algorithm.

Our second contribution is of independent interest as it provides an alternative but more flexible view on May-Ozerov Near Neighbour algorithm for the Hamming metric. We give simple formulas for analysing its complexity which allow us to stay in the Hamming space, i.e., without reductions from other metrics as it is usually done in the similarity search literature [Chr17]. The third contribution answers the problem left open in [KT17], namely, how to use Near Neighbour technique within quantum walks. Our results are summarized below.

Table 1. Running time and space complexities of ISD algorithms (full distance decoding). The columns give the exponent-constants c , i.e., runtime and memory complexities are of the form $\mathcal{O}(2^{cn})$. For Prange’s algorithm, the space is $\text{poly}(n)$.

Algorithm	Quantum		Classical	
	Time	Space	Time	Space
Prange [Ber10, Pra62]	0.060350	–	0.120600	–
Stern/Dumer [Ste89, Dum91]				
+ Shamir-Schroepel (SS) [KT17]	0.059697	0.00618	0.116035	0.03644
+ Near Neighbour (NN) Sect. 4	0.059922	0.00897	0.113762	0.04248
+ SS + NN Sect. 4	0.059450	0.00808		
MMT [MMT11]				
– Kachigar-Tillich [KT17]	0.059037	0.01502	0.111468	0.05408
– Section 3	0.058059	0.01849		
BJMM [BJMM12]				
– Kachigar-Tillich [KT17]	0.058696	0.01877	0.101998	0.07590
– Section 3	0.058040	0.01866		

For each classical algorithm, Table 1 gives running times and space complexities of their quantum counterparts. This work improves over Kachigar-Tillich quantum versions of MMT in time, and of BJMM in both time and space. The

improvement comes from different analysis: as explained in Sect. 2, the (asymptotical) complexity of these two algorithms is the maximum between the runtimes of several subroutines. The authors in [KT17] assume that the maximum is attained at a certain subroutine and, during the derivation of the algorithm's parameters, enforce that the runtimes of all other subroutines remain below. On the contrary, we assume that another subroutine is dominant. By the 'quantum space' in Table 1, we mean the number of qubits in a quantum state an algorithm operates on. The figures from Table 1 are obtained using the optimization package implemented in Maple. The program is available at <http://perso.ens-lyon.fr/elena.kirshanova/>.

In Sect. 4 we show how to combine the Near Neighbour search of May and Ozerov [MO15] with quantum version of the ISD algorithm due to Dumer [Dum91]. By itself, this does not improve over the algorithm given in [KT17]. But, combined with the so-called Shamir-Schroepel trick [SS81] already used in [KT17], we can slightly improve the runtime complexity of the algorithm. We note that, as in the classical setting, the Near Neighbour technique requires more memory, but we are still far from the Time = Memory regime. It turns out that, as opposed to the classical case, quantum Near Neighbour search does not improve MMT or BJMM. We argue why this is the case at the end of Sect. 4.

2 Preliminaries

We give an overview on classical algorithms for ISD, namely, Prange [Pra62], Stern and its variants [Ste89, Dum91], MMT [MMT11], and BJMM [BJMM12] algorithms. We continue with known quantum speed-ups for these algorithms.

2.1 Classical ISD Algorithms

All known ISD algorithms try to find the error-vector \mathbf{e} by a clever enumeration of the search space for \mathbf{e} , which is of size $\binom{n}{w} \approx 2^{n \cdot H(\frac{w}{n})}$, where $H(x) = -x \log x - (1-x) \log(1-x)$ is the binary entropy function. In the analysis of ISD algorithms, it is common to relate the parameters w (the error-weight), and k (the rank of a code) to dimension n , and simplify the running times to the form 2^{cn} for some constant c .¹ To do this, we make use of Gilbert-Varshamov bound which states that $\frac{k}{n} = 1 - H(\frac{w}{n})$ as $n \rightarrow \infty$. This gives us a way to express w as a function of n and k . Finally, the running time of an ISD algorithm is obtained by a brute-force search over all $\frac{k}{n} \in [0, \frac{1}{2}]$ (up to some precision) that leads to the worst-case complexity. In the classical setting, this worst-case is reached by codes of rate $\frac{k}{n} \approx 0.447$, while in the quantum regime it is $\frac{k}{n} \approx 0.45$.

Decoding algorithms start by permuting the columns of \mathbf{H} which is equivalent to permuting the positions of 1's in \mathbf{e} . The goal is to find a permutation $\pi \in S_n$

¹ We omit sub-exponential in n factors throughout, because we are only interested in the constant c . Furthermore, our analysis is for an average case and we sometimes omit the word 'expected'.

such that $\pi(\mathbf{e})$ has exactly $p \leq w$ 1's on the first k coordinates and the remaining weight of $w - p$ is distributed over the last $n - k$ coordinates. All known ISD algorithms make use of the fact that such a permutation is found. We expect to find a good π after $\mathcal{P}(p)$ trials, where

$$\mathcal{P}(p) = \frac{\binom{k}{p} \binom{n-k}{w-p}}{\binom{n}{w}}. \tag{1}$$

The choice of p and how we proceed with $\pi(\mathbf{H})$ depends on the ISD algorithm.

For example, *Prange's* algorithm [Pra62] searches for a permutation π that leads to $p = 0$. To check whether a candidate π is good, it transforms $\pi(\mathbf{H})$ into systematic form $[\mathbf{Q} \mid \mathbf{I}_{n-k}]$ (provided the last $n - k$ columns of $\pi(\mathbf{H})$ form an invertible matrix which happens with constant success probability). The same transformation is applied to the syndrome \mathbf{s} giving a new syndrome $\bar{\mathbf{s}}$. From the choice of p , it is easy to see that for a good π , we just 'read-off' the error-vector from the new syndrome, i.e., $\pi(\mathbf{e}) = \bar{\mathbf{s}}$, and to verify a candidate π , we check if $wt(\bar{\mathbf{s}}) = w$. We expect to find a good π after $\mathcal{P}(0)$ trials.

From now on, we assume that we work with systematic form of \mathbf{H} , i.e.

$$[\mathbf{Q} \mid \mathbf{I}_{n-k}] \cdot \mathbf{e} = \bar{\mathbf{s}} \quad \text{for } \mathbf{Q} \in \mathbb{F}_2^{n-k \times k}. \tag{2}$$

Other than restricting the weight of \mathbf{e} to be 0 on the last $n - k$ coordinates, we may as well allow $p > 0$ at the price of a more expensive check for π . This is the choice of *Stern's* algorithm [Ste89], which was later improved in [Dum91]. We describe the improved version. We start by adjusting the systematic form of \mathbf{H} introducing the ℓ -length 0-window, so that Eq. 2 becomes

$$\left[\mathbf{Q} \mid \frac{\mathbf{0}}{\mathbf{I}_{n-k-\ell}} \right] \cdot \mathbf{e} = \bar{\mathbf{s}} \quad \text{for } \mathbf{Q} \in \mathbb{F}_2^{n-k \times k+\ell}. \tag{3}$$

Now we search for a permutation π that splits the error as

$$\mathbf{e} = [\mathbf{e}_1 \mid \mathbf{0}^{\frac{k+\ell}{2}} \mid \mathbf{0}^{n-k-\ell}] + [\mathbf{0}^{\frac{k+\ell}{2}} \mid \mathbf{e}_2 \mid \mathbf{0}^{n-k}] + [\mathbf{0}^{k+\ell} \mid \mathbf{e}_3],$$

such that $wt(\mathbf{e}_1) = wt(\mathbf{e}_2) = p/2$ and $wt(\mathbf{e}_3) = w - p$, where \mathbf{e}_i 's are of appropriate dimensions. With such an \mathbf{e} , we can re-write Eq. (3) as

$$\mathbf{Q} \cdot [\mathbf{e}_1 \mid \mathbf{0}^{\frac{k+\ell}{2}}] + \mathbf{Q} \cdot [\mathbf{0}^{\frac{k+\ell}{2}} \mid \mathbf{e}_2] = \bar{\mathbf{s}} + \mathbf{e}_3. \tag{4}$$

We enumerate all possible $\binom{(k+\ell)/2}{p/2}$ vectors of the form $\mathbf{v}_1 := \mathbf{Q}[\mathbf{e}_1 \mid \mathbf{0}^{\frac{k+\ell}{2}}]$ into a list \mathcal{L}_1 and all vectors of the form $\mathbf{v}_2 := \mathbf{Q}[\mathbf{0}^{\frac{k+\ell}{2}} \mid \mathbf{e}_2] + \bar{\mathbf{s}}$ into a list \mathcal{L}_2 . The above equation tells us that for the correct pair $(\mathbf{e}_1, \mathbf{e}_2)$, the sum of the corresponding list-vectors equals to $\mathbf{0}$ on the first ℓ -coordinates. We search for two vectors $\mathbf{v}_1 \in \mathcal{L}_1, \mathbf{v}_2 \in \mathcal{L}_2$ that are equal on this ℓ -window. We call such pair $(\mathbf{v}_1, \mathbf{v}_2)$ a *match*. We check among these matches if the Hamming distance between $\mathbf{v}_1, \mathbf{v}_2$, denoted $dt(\mathbf{v}_1, \mathbf{v}_2)$, is $w - p$. To retrieve the error-vector, we store \mathbf{e}_i 's together with the corresponding \mathbf{v}_i 's in the lists. The probability of finding a permutation that meets all the requirements is

$$\mathcal{P}(p, \ell) = \frac{\binom{k+\ell}{p} \binom{n-k-\ell}{w-p}}{\binom{n}{w}}. \tag{5}$$

It would be more precise to have $\left(\frac{k+\ell}{p/2}\right)^2$ instead of $\binom{k+\ell}{p}$ in the above formula, but these two quantities differ by only a factor of $\text{poly}(n)$ which we ignore. The expected running time of the algorithm is then

$$T = \mathcal{P}(p, \ell)^{-1} \cdot \max \left\{ |\mathcal{L}_2|, \frac{|\mathcal{L}_1| \cdot |\mathcal{L}_2|}{2^\ell} \right\}, \tag{6}$$

where the first argument of \max is the time to sort \mathcal{L}_2 , the second is the expected number of pairs from $\mathcal{L}_1 \times \mathcal{L}_2$ that are equal on ℓ , which we check for a solution. See Fig. 1 for an illustration of the algorithm.

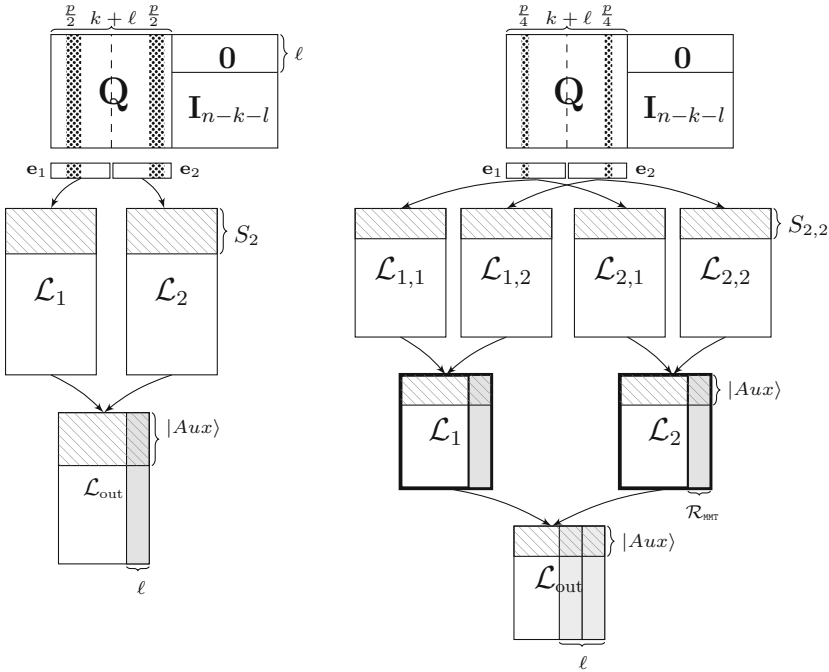


Fig. 1. *On the left:* A variant of Stern’s ISD algorithm due to Dumer [Dum91]. The list \mathcal{L}_1 is constructed from all possible $p/2$ -weight vectors $\mathbf{e}_1 \in \mathbb{F}_2^{(k+\ell)/2}$: $\mathcal{L}_1 = \{(\mathbf{e}_1, \mathbf{Q} \cdot [\mathbf{e}_1 || \mathbf{0}^{(k+\ell)/2}])\}$. \mathcal{L}_2 is constructed similarly with $\mathbf{0}^{(k+\ell)/2}$ and \mathbf{e}_1 swapped. Gray-shaded vertical strip indicates the coordinates on which the elements $\mathbf{v}_1 \in \mathcal{L}_1$ and $\mathbf{v}_2 \in \mathcal{L}_2$ match. Line-shaded horizontal strips indicate a subset of lists stored on quantum registers during the execution of quantum walk search algorithm.

On the right: May-Meurer-Thomae decoding [MMT11]. The lists $\mathcal{L}_1, \mathcal{L}_2$ are shorter than in Dumer’s algorithm as their elements already match on \mathcal{R}_{MMT} -coordinates. It is the creation of the lists $\mathcal{L}_1, \mathcal{L}_2$ (indicated in bold) that dominates over the creation of $\mathcal{L}_{i,j}, i, j \in \{1, 2\}$ and of \mathcal{L}_{out} and hence, determines the running time of the algorithm.

The *Representation technique* of [BJMM12, MMT11] further improves the search for matching vectors by constructing the lists $\mathcal{L}_1, \mathcal{L}_2$ faster. Now the list \mathcal{L}_1 consists of all pairs $(\mathbf{e}_1, \mathbf{Q}\mathbf{e}_1)$ where $\mathbf{e}_1 \in \mathbb{F}_2^{k+\ell}$ (as opposed to $\mathbf{e}_1 \in \mathbb{F}_2^{(k+\ell)/2}$) with $wt(\mathbf{e}_1) = p/2$. Similarly, $\mathcal{L}_2 = \{(\mathbf{e}_2, \mathbf{Q}\mathbf{e}_2) \mid \forall \mathbf{e}_2 \in \mathbb{F}_2^{k+\ell}, wt(\mathbf{e}_2) = p/2\}$. The key observation is that now there are $\mathcal{R}_{\text{MMT}} := \binom{p}{p/2}$ ways to represent the target \mathbf{e} as $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$. Hence, it is enough to construct only an \mathcal{R}_{MMT} -fraction of $\mathcal{L}_1, \mathcal{L}_2$. Such a fraction of \mathcal{L}_1 (analogously, for \mathcal{L}_2) is built by merging in the meet-in-the-middle way yet another two lists $\mathcal{L}_{1,1}$ and $\mathcal{L}_{1,2}$ filled with vectors of the form $\mathbf{Q}[\mathbf{e}_{1,1} \parallel \mathbf{0}^{(k+\ell)/2}]$ (for $\mathcal{L}_{1,1}$) and $\mathbf{Q}[\mathbf{0}^{(k+\ell)/2} \parallel \mathbf{e}_{1,2}]$ (for $\mathcal{L}_{1,2}$) for all $p/4$ -weight $\mathbf{e}_{1,1}$ and $\mathbf{e}_{1,2}$, respectively. These starting lists are of size

$$|L_{i,j}| = \binom{(k+\ell)/2}{p/4}, \quad i, j \in \{1, 2\}. \tag{7}$$

During the merge, we force vectors from $\mathcal{L}_{1,1}$ be equal to vectors from $\mathcal{L}_{1,2}$ on $\log \mathcal{R}_{\text{MMT}}$ coordinates leaving only one (in expectation) pair $(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ whose sum gives \mathbf{e} (see Fig. 1, right). Here and later, we shall abuse notations slightly: technically, the list elements are pairs $(\mathbf{e}, \mathbf{Q}\mathbf{e})$, but the merge is always done on the second element, and the error retrieval is done on the first.

The number of necessary permutations we need to try is given by Eq. (5). Provided a good π is found, the time to find the correct \mathbf{e} is now given by $\max\{|L_{1,1}|, |L_{1,1}|^2/2^{\mathcal{R}_{\text{MMT}}}, |L_{1,1}|^4/2^{\ell+\mathcal{R}_{\text{MMT}}}\}$. This is the maximum between (I) the size of starting lists, (II) the size of the output after the first merge on $\log \mathcal{R}_{\text{MMT}}$ coordinates, and (III) the size of the final output after merging on the remaining $\ell - \log \mathcal{R}_{\text{MMT}}$ coordinates. Optimization for p, ℓ reveals that (II) is the maximum. Overall, the expected complexity of MMT is

$$T_{\text{MMT}} = \mathcal{P}(p, \ell) \cdot \frac{|L_{i,j}|^2}{\mathcal{R}_{\text{MMT}}}. \tag{8}$$

Becker-Jeux-May-Meurer in [BJMM12] further improves the merging step (i.e., the dominant one) noticing that zero-coordinates of \mathbf{e} can be split in $\mathbf{e}_1, \mathbf{e}_2$ not only as $0+0$, but also as $1+1$. It turns out that constructing longer starting lists $\mathcal{L}_{i,j}$ using \mathbf{e}_i of weights $wt(\mathbf{e}_i) = p/2 + \varepsilon$ is profitable as it significantly increases the number of representations from $\binom{p}{p/2}$ to $\mathcal{R}_{\text{BJMM}} := \binom{p}{p/2} \binom{k+\ell-p}{\varepsilon}$, thus allowing a better balance between the two merges: the first merge on $\log \mathcal{R}_{\text{BJMM}}$ coordinates and the second on $\ell - \log \mathcal{R}_{\text{BJMM}}$ coordinates. The expected running time of the BJMM algorithm is given by

$$T_{\text{BJMM}} = \mathcal{P}(p, \ell) \cdot \frac{|L_{i,j}|^2}{\mathcal{R}_{\text{BJMM}}}, \quad \text{where } |L_{i,j}| = \binom{(k+\ell)/2}{p/4 + \varepsilon}. \tag{9}$$

In fact, the actual BJMM algorithm is slightly more complicated than we have described, but the main contribution comes from adding ‘ $1+1$ ’ to representations, so hereafter we refer to this simplified version as BJMM.

2.2 Quantum ISD Algorithms

Quantum ISD using Grover’s algorithm. To speed-up Prange’s algorithm, Bernstein in [Ber10] uses Grover’s search over the space of permutations, which is of size $\mathcal{P}(0) = \binom{w}{k} / \binom{n}{w}$. This drops the expected runtime from $2^{0.1206n}$ (classical) down to $2^{0.06035n}$ (quantum), cf. Table 1. The approach has an advantage over all the quantum algorithms we discuss later as it requires quantum registers to store data of only poly(n) size. To obtain a quantum speed-up for other ISD algorithms like Stern’s, MMT, BJMM, we need to describe quantum walks.

Quantum walks. At the heart of the above ISD algorithms (except Prange’s) is the search for vectors from given lists that satisfy a certain relation. This task can be generalized to the k -list matching problem.

Definition 1 (k -list matching problem). *Let k be fixed. Given k equal sized lists $\mathcal{L}_1, \dots, \mathcal{L}_k$ of binary vectors and a function g that decides whether a k -tuple $(\mathbf{v}_1, \dots, \mathbf{v}_k) \in \mathcal{L}_1 \times \dots \times \mathcal{L}_k$ forms a ‘match’ or not (outputs 1 in case of a ‘match’), find all k -tuples $(\mathbf{v}_1, \dots, \mathbf{v}_k) \in \mathcal{L}_1 \times \dots \times \mathcal{L}_k$ s.t. $g(\mathbf{v}_1, \dots, \mathbf{v}_k) = 1$.*

For example, the Stern’s algorithm uses $k = 2$ and its g decides for a ‘match’ whenever a pair $(\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ is equal on certain fixed ℓ coordinates. For MMT or BJMM, we deal with four lists $\mathcal{L}_1, \dots, \mathcal{L}_4$, and function g decides for the match if $\mathbf{v}_1 + \mathbf{v}_2, \mathbf{v}_3 + \mathbf{v}_4$ are equal on a certain part of coordinates (merge of \mathcal{L}_1 with \mathcal{L}_2 , and \mathcal{L}_3 with \mathcal{L}_4) and, in addition, $\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3 + \mathbf{v}_4$ is 0 on ℓ .

Quantumly we solve the above problem with the algorithm of Ambainis [Amb04]. Originally it was described only for the case $k = 2$ (search version of the so-called *Element distinctness* problem), but later extended to a more general setting, [CE05]. We note that the complexity analysis in [CE05] is done in terms of *query* calls to the g function, while here we take into account the actual time to compute g . Ambainis algorithm is best described as a quantum walk on the so-called Johnson Graph.

Definition 2 (Johnson graph and its eigenvalue gap). *The Johnson graph $J(N, r)$ for an N -size list is an undirected graph with vertices labelled by all r -size subsets of the list, and with an edge between two vertices S, S' iff $|S \cap S'| = r - 1$. It follows that $J(N, r)$ has $\binom{N}{r}$ vertices. Its eigenvalue gap is $\delta = \frac{N}{r(N-r)}$, [BCA89].*

Let us briefly explain how we solve the k -list matching problem using quantum walks. Our description follows the so-called MNRS framework [MNRS11] due to Magniez-Nayak-Roland-Santha, which measures the complexity of a quantum walk search algorithm in the costs of their Setup, Update, and Check phases.

To setup the walk, we first prepare a uniform superposition over all r -size subsets $S_i \subset \mathcal{L}_i$ together with an auxiliary register (normalization omitted):

$$\sum_{S_i \subset \mathcal{L}_i, |S_i|=r} |S_1\rangle \otimes \dots \otimes |S_k\rangle \otimes |Aux\rangle.$$

The auxiliary register $|Aux\rangle$ contains all the information needed to decide whether S_1, \dots, S_k contains a match. In the ISD setting, $|Aux\rangle$ stores intermediate and output lists of the matching process. For example, in Stern’s algorithm ($k = 2$) $|Aux\rangle$ contains all pairs $(\mathbf{v}_1, \mathbf{v}_2) \in S_1 \times S_2$ that match on ℓ coordinates. In case the merge is done in several steps like in MTT ($k = 4$), the intermediate sublists are also stored in $|Aux\rangle$ (see Fig. 1).

The running time and the space complexity of the Setup phase are essentially the running time and the space complexity of the corresponding ISD algorithm with the input lists of size $|S_i| = r$ instead of $|\mathcal{L}_i|$. By the end of the Setup phase, we have a superposition over all r -sublists S_1, \dots, S_k of $\mathcal{L}_1, \dots, \mathcal{L}_k$, where each (S_1, \dots, S_k) is entangled with the register $|Aux\rangle$ that contains the result of merging (S_1, \dots, S_k) into $S_{\text{out}} \subset \mathcal{L}_{\text{out}}$. Also, during the creation of S_{out} we can already tell if it contains the error-vector \mathbf{e} that solves the ISD problem. When we talk about quantum space of an ISD algorithm (e.g., Table 1), we mean the size of the $|Aux\rangle$ register.

Next, in the Update phase we choose a sublist S_i and replace one element $\mathbf{v}_i \in S_i$ by $\mathbf{v}'_i \notin S_i$. This is one step of a walk on the Johnson graph. We update the data stored in $|Aux\rangle$: remove all the pairs in the merged lists that involve \mathbf{v}_i and create possibly new matches with \mathbf{v}'_i . We assume the sub-lists S_i ’s are kept sorted and stored in a data-structure that allows fast insertions/removals (e.g., radix trees as proposed in [BJLM13]). We also assume that elements in S_1, \dots, S_k that result in a match, store pointers to their match. For example, in Stern’s algorithm $\mathbf{v}_1 \in S_1, \mathbf{v}_2 \in S_2$ give a match, we keep a pointer to $\mathbf{v}_1 + \mathbf{v}_2 \in S_{\text{out}}$ and also a pointer from $\mathbf{v}_1 + \mathbf{v}_2$ to $\mathbf{v}_1 \in S_1, \mathbf{v}_2 \in S_2$.

After we have performed $\Theta(1/\sqrt{\delta})$ updates (recall, δ is the eigenvalue gap of $J(N, r)$), we check if the updated register $|S_1\rangle \otimes \dots \otimes |S_k\rangle \otimes |Aux\rangle$ gives a match. This is the Checking phase.

Thanks to the MNRS framework, once we know the costs of (a) the Setup phase T_S , (b) the Update phase T_U , and (c) the Checking phase T_C , we know that after T_{QW} many steps, we measure a register $|S_1\rangle \otimes \dots \otimes |S_k\rangle \otimes |Aux\rangle$ that contains the correct error-vector with overwhelming probability, where

$$T_{\text{QW}} = T_S + \frac{1}{\sqrt{\varepsilon}} \left(\frac{1}{\sqrt{\delta}} \cdot T_U + T_C \right). \tag{10}$$

In the above formula, ε is a fraction of vertices in $J(N, r)$ that contain the correct error-vector. For a fixed k , we have $\varepsilon \approx r^k/N^k$ where $N = |\mathcal{L}_1| = \dots = |\mathcal{L}_k|$. Strictly speaking, the walk we have just described is a walk on a k -Cartesian product of Johnson graphs – one for each sublist S_i , so the value δ in Eq. (10) must be the eigenvalue gap for such a large graph. As proved in [KT17, Theorem 2], for fixed constant k , it is lower-bounded by $\frac{N}{k \cdot r(N-r)}$. The analysis of [KT17] as well ours are asymptotical, so we ignore the constant factor of $1/k$. An optimal choice for r that minimizes Eq. (10) is discussed in the next section.

Kachigar-Tillich quantum ISD algorithms. The quantum walk search algorithm described above solves the ISD problem provided we have found a

permutation π that gives the desired distribution of 1's in the error-vector. Kachigar and Tillich in [KT17] suggest to run Grover's algorithm for π with the 'checking' function for Grover's search being a routine for the k -list matching problem. Their ISD algorithm performs transformations on the quantum state of the form (normalization omitted):

$$\sum_{i=1}^{\mathcal{P}(p,\ell)} |\pi_i\rangle |\pi_i(\mathbf{H})\rangle \otimes \underbrace{\left[\sum_{S_i \subset \mathcal{L}_i, |S_i|=r} |S_1\rangle \otimes \dots \otimes |S_k\rangle \otimes |Aux\rangle \right]}_{\text{Quantum Walk = Check for the outer Grover}} \otimes |\text{Is } \pi \text{ good?}\rangle \quad (11)$$

The outer-search is Grover's algorithm over $\mathcal{P}(p, \ell)$ permutations, where $\mathcal{P}(p, \ell)$ is chosen such that we expect to have one π that leads to a good permutation of 1's in the error-vector (see Eq. (5)). The check if a permutation π is good is realized via quantum walk search for k vectors $\mathbf{v}_1, \dots, \mathbf{v}_k \in S_1 \times \dots \times S_k$ that match on certain coordinates and lead to the correct error vector. Note an important difference between classical and quantum settings: during the quantum walk we search over sublists $S_i \subset \mathcal{L}_i$ which are exponentially shorter than \mathcal{L}_i .

After T_{QW} steps, the register $|Aux\rangle$ contains a k -tuple $(\mathbf{v}_1, \dots, \mathbf{v}_k)$ that leads to the correct error vector provided a permutation π is good. Hence, after $\tilde{O}(\sqrt{\mathcal{P}(p, \ell)} \cdot T_{\text{QW}})$ steps, the measurement of the first register gives a good π with constant success probability. The resulting state will be entangled with registers that store S_1, \dots, S_k together with the pointers to the matching elements. Once we measure S_1, \dots, S_k , we retrieve these pointers and, finally, reconstruct the error vector as in the classical case.

Quantum Shamir-Schroeppeel technique was introduced in [SS81] to reduce the memory complexity of a generic meet-in-the-middle attack, i.e., the k -list matching problem for $k = 2$. Assume we want to find a pair $\mathbf{v}_1, \mathbf{v}_2 \in \mathcal{L}_1 \times \mathcal{L}_2$ s.t. $\mathbf{v}_1 = \mathbf{v}_2$ on certain ℓ coordinates. Assume further that we can decompose $\mathcal{L}_1 = \mathcal{L}_{1,1} + \mathcal{L}_{1,2}$ s.t. $|\mathcal{L}_{1,1}| = |\mathcal{L}_{1,2}| = \sqrt{|\mathcal{L}_1|}$ (analogously, for \mathcal{L}_2). The idea of Shamir and Schroeppeel is to guess that the correct vectors $\mathbf{v}_1, \mathbf{v}_2$ are equal to some $\mathbf{t} \in \mathbb{F}_2^{\ell'}$ on $\ell' \leq \ell$ coordinates and enumerate all such pairs. Namely, we enumerate \mathbf{v}_1 by constructing \mathcal{L}_1 in the meet-in-the-middle way from $\mathcal{L}_{1,1}, \mathcal{L}_{1,2}$ in time $\max\{\sqrt{|\mathcal{L}_1|}, |\mathcal{L}_1|/2^{\ell'}\}$, s.t. \mathcal{L}_1 only contains vectors that are equal to \mathbf{t} on ℓ' (same for \mathcal{L}_2). Classically, we make $2^{\ell'}$ guesses for \mathbf{t} , so the overall time of the algorithm will be $|\mathcal{L}_1|$ (same as naive 2-list matching), but we save in memory.

In [KT17], in order to improve not only in memory, but also in time, the authors run Grover's search over $2^{\ell'}$ guesses for \mathbf{t} . Indeed, this gives a speed-up for ISD algorithms that solve the 2-list matching problem (cf. the complexities of Dumer's algorithm in Table 1). However, as we argue in the next section, for ISD algorithms that operate on 4 (or more) lists and whose complexity is *not* dominated by the creation of the starting lists (like MMT or BJMM), quantum Shamir-Schroeppeel technique does not bring an advantage.

3 Quantum MMT and BJMM Algorithms

In this section we analyse the complexity of quantum versions of MMT and BJMM ISD algorithms and argue that the analysis in [KT17] is slightly sub-optimal when optimized for time. We note that the way we apply and analyse quantum walks to ISD closely resembles Bernstein’s et al. algorithm for the Subset Sum problem [BJLM13].

Let us first look at the generalized version of a quantum ISD algorithm, where we can plug-in any of the ISD algorithms described in Sect. 2. Recall that on input, we receive $(\mathbf{H}, \mathbf{s}) \in \mathbb{F}_2^{(n-k) \times n} \times \mathbb{F}_2^{n-k}$, and are asked to output $\mathbf{e} \in \mathbb{F}_2^n$ of weight $wt(\mathbf{e}) = w$ that satisfies $\mathbf{H}\mathbf{e}^t = \mathbf{s}$. Algorithm 1 below can be viewed as a ‘meta’ quantum algorithm for ISD.

Algorithm 1. A quantum ISD algorithm

- 1: Prepare a superposition over \mathcal{P} -many permutations π
 - 2: For each π
 - a: Setup a superposition $|S_1\rangle \otimes \dots \otimes |S_k\rangle \otimes |Aux\rangle$ for $S_i \subset \mathcal{L}_i, |S_i| = r$
 - b: Run a quantum walk search on $|S_1\rangle \otimes \dots \otimes |S_k\rangle \otimes |Aux\rangle$ to find a matching tuple $(\mathbf{e}_1, \dots, \mathbf{e}_k) \in S_1 \times \dots \times S_k$, if exists; indicate otherwise that no tuple is found.
 - 3: Apply amplitude amplification (Grover’s search) on Step 1 for those π that led to a match on Step 2.b. Measure the register π and then the register $|Aux\rangle$.
-

The algorithm is parametrized by (I) the size of the permutation space \mathcal{P} we iterate over in order to find the desired distribution of 1’s in the solution (e.g., Eq. (5) for MMT); (II) k – the number of staring lists \mathcal{L}_i ’s an ISD-algorithm considers (e.g., $k = 0$ for Prange, $k = 2$ for Stern/Dumer, $k = 4$ for MMT); (III) r – the size of S_i ’s, $1 \leq i \leq k$. The asymptotic complexity of Algorithm 1 will depend on these quantities as we now explain in detail.

Step 1 consists in preparing a superposition $\sum_{i=1}^{\mathcal{P}} |\pi_i\rangle |\pi_i(\mathbf{H})\rangle$, which is efficient. Step 2 is a quantum walk algorithm for the k -list matching problem, i.e. search for all $(\mathbf{e}_1, \dots, \mathbf{e}_k) \in \mathcal{L}_1 \times \dots \times \mathcal{L}_k$ from which the solution vector can be constructed. The cost of Step 2 can be split into the cost of the Setup phase (Step 2(a)) and the cost of the Update and Check phases (Step 2(b)).

The cost of the Step 2(a) – preparing a superposition over k -tensor product of $S_i \subset \mathcal{L}_i$ and computing the data for $|Aux\rangle$ – is essentially the cost of a classical ISD algorithm, where on input instead of the lists \mathcal{L}_i ’s, we consider sublists S_i of size $r \ll |\mathcal{L}_i|$. Recall that ‘computing the data for $|Aux\rangle$ ’ means constructing the subset $S_{out} \subset \mathcal{L}_{out}$ using only elements from S_i ’s (see Fig. 1). Step 2(b) performs a quantum walk over the k -Cartesian product of Johnson Graphs, $J(|L_i|, r)^{\otimes k}$, with eigenvalue gap $\delta = \Theta(|L_i|/(r \cdot (|L_i| - r))) \approx 1/r$ for $r \ll |L_i|$. To estimate ε – the fraction of (S_1, \dots, S_k) that give the solution, note that with probability $\Theta(r/|L_i|)$, an r -size subset S_i contains an element \mathbf{e}_i that contributes to the solution. Hence, k such subsets – one vertex of $J(|L_i|, r)^{\otimes k}$ –

contain the solution $(\mathbf{e}_1, \dots, \mathbf{e}_k)$ with probability $\varepsilon = (r/|L_i|)^k$. Once we know the time to create $|Aux\rangle$, δ and ε , we obtain the complexity of Step 2 from Eq. (10).

Finally, Grover's search over \mathcal{P} -many permutations requires $\sqrt{\mathcal{P}}$ calls to a 'checking' function s.t. a measurement will output the good π . The measurement will collapse the state given in Eq. (11) into a superposition of $|S_1\rangle \otimes \dots \otimes |S_k\rangle \otimes |Aux\rangle$, where the amplitude of those $|Aux\rangle$ that contain the actual solution \mathbf{e} will be amplified. Measurement of $|Aux\rangle$ leads to the solution. Regarding Step 2 as 'checking' routine for amplitude-amplification of Step 1 and assuming that an ISD algorithm on input-lists of size r has classical running time $T_{\text{ISD}}(r)$, we obtain the following complexity of Algorithm 1:

Theorem 1. *Assume we run Algorithm 1 with a classical ISD algorithm that (I) expects after \mathcal{P} permutations of the columns of \mathbf{H} to find the desired weight-distribution for the error-vector, (II) performs the search for the error-vector over k lists each of size $|L|$ in (classical) time $T_{\text{ISD}}(|L|)$. Then for $r \ll |\mathcal{L}|$ satisfying $T_{\text{ISD}}(r) = \tilde{\mathcal{O}}\left(\sqrt{\frac{|\mathcal{L}|^k}{r^{k+1}}}\right)$, the running time of Algorithm 1 is*

$$T_{\text{ISD}_q} = \sqrt{\mathcal{P}} \cdot T_{\text{ISD}}(r).$$

In particular, the MMT algorithm [MMT11] has $k = 4$, $\mathcal{P} = \mathcal{P}(p, \ell)$ given in Eq. (5), $|\mathcal{L}|$ given in Eq. (7), and $\mathcal{R}_{\text{MMT}} := \binom{p}{p/2}$, leading to

$$T_{\text{MMT}_q} = \tilde{\mathcal{O}}\left(\sqrt{\mathcal{P}(p, \ell)} \cdot \frac{|\mathcal{L}|^{\frac{8}{7}}}{\mathcal{R}_{\text{MMT}}^{\frac{3}{7}}}\right).$$

Similarly, for the BJMM algorithm [BJMM12] with starting lists-sizes $|\mathcal{L}|$ given in Eq. (9) and $\mathcal{R}_{\text{BJMM}} := \binom{p}{p/2} \binom{k+\ell-p}{\varepsilon}$, we have

$$T_{\text{BJMM}_q} = \tilde{\mathcal{O}}\left(\sqrt{\mathcal{P}(p, \ell)} \cdot \frac{|\mathcal{L}|^{\frac{8}{7}}}{\mathcal{R}_{\text{BJMM}}^{\frac{3}{7}}}\right).$$

Proof. The first statement follows from the above discussion: Grover's search for a good π makes $\sqrt{\mathcal{P}}$ 'calls', where each 'call' is a quantum walk search of complexity $T_{\text{ISD}}(r)$. The condition on r is set such that Steps 2(a) and 2(b) in Algorithm 1 are asymptotically balanced, namely, we want $T_{\text{Setup}} = \frac{1}{\sqrt{\varepsilon}} \cdot \frac{1}{\sqrt{\delta}} \cdot \tilde{\mathcal{O}}(\log r)$, cf. Eq. (10). We remind that S_i 's are stored in a data-structure that makes the Update and Check of complexity $\tilde{\mathcal{O}}(\log r)$. Since $T_{\text{Setup}} = T_{\text{ISD}}(r)$, $\delta \approx 1/r$, and $\varepsilon = (r/|L_i|)^k$, the optimal choice for r should satisfy $T_{\text{ISD}}(r) = \tilde{\mathcal{O}}(\sqrt{|\mathcal{L}|^k/r^{k+1}})$.

For the MMT algorithm, the dominating step is the construction of the lists $\mathcal{L}_1, \mathcal{L}_2$ whose elements are already equal on a certain number of coordinates denoted $\log \mathcal{R}_{\text{MMT}} \approx p$ in Sect. 2 (see Fig. 1, right). This step is of complexity $T_{\text{MMT}}(|\mathcal{L}|) = \tilde{\mathcal{O}}(|\mathcal{L}|^2/\mathcal{R}_{\text{MMT}})$, where $|\mathcal{L}|$ is the size of the starting lists (the four upper-most lists on Fig. 1). Solving $T_{\text{MMT}}(r) \stackrel{!}{=} \sqrt{\frac{|\mathcal{L}|^k}{r^{k+1}}}$ for r , we receive

$r = |\mathcal{L}|^{\frac{4}{7}} \cdot 2^{\frac{2}{7}p}$ as the optimal size for S_i 's. Hence, the running time of the quantum walk for MMT is $T_{\text{MMT}}(r) = \tilde{O}(|\mathcal{L}|^{\frac{8}{7}}/\mathcal{R}_{\text{MMT}}^{\frac{3}{7}}) = \tilde{O}(|\mathcal{L}|^{\frac{8}{7}}/2^{\frac{3}{7}p})$.

The BJMM algorithm differs from MMT in the number of representations $\mathcal{R}_{\text{BJMM}}$ and the size of the starting lists $|\mathcal{L}|$. Similar to MMT, we choose $r = |\mathcal{L}|^{\frac{4}{7}} \cdot \mathcal{R}_{\text{BJMM}}^{\frac{2}{7}}$, the complexity of the quantum walk for BJMM becomes $\tilde{O}(|\mathcal{L}|^{\frac{8}{7}}/\mathcal{R}_{\text{BJMM}}^{\frac{3}{7}})$. \square

The analysis of [KT17] differs from ours in computing $T_{\text{ISD}}(r)$: in [KT17] it is assumed that the dominant part MMT or BJMM algorithm is the first merge (i.e., merging the starting lists on $\log \mathcal{R}_{\text{MMT}}$ or $\log \mathcal{R}_{\text{BJMM}}$ coordinates, which results in choosing $r = |\mathcal{L}|^{4/5}$). This is why they used the so-called Shamir-Schroepfel trick [SS81]: reduce the size of the starting lists and repeat all the merges many times with the new starting lists of smaller size. Classically, this trick allows to reduce only space, while quantumly one could also hope to improve time using Grover over repetitions of merges. Our optimization, however, shows that Shamir-Schroepfel trick does not give any time improvement as it is the case classically, because there is no reason to optimize a non-dominant step. Again, it helped the analysis of [KT17], but this analysis assumes a non-optimal choice of parameters that makes the first merging step dominant. One can, however, using this idea, save in memory for the MMT algorithm (cf. Table 1)

Classically, the improvements over Prange achieved by recent algorithms are quite substantial: BJMM gains a factor of $2^{0.019 \cdot n}$ in the leading-order term. Quantumly, however, the improvement is less pronounced. The reason lies in the fact that the speed-up coming from Grover's search is much larger than the speed-up offered by the quantum walk. Also, the k -list matching problem become harder (quantumly) once we increase k (the fraction of 'good' subsets ε becomes smaller).

4 Decoding with Near Neighbour

For a reader familiar with Indyk-Motwani locality-sensitive hashing [IM98] for Near Neighbour search (defined below), Stern's algorithm and its improvements [Dum91] essentially implement such hashing by projecting on ℓ -coordinates and applying it to the lists $\mathcal{L}_1, \mathcal{L}_2$. In this section, we consider another Near Neighbour technique.

4.1 Re-interpretation of May-Ozerov Near Neighbour Algorithm

The best known classical ISD algorithm is due to May-Ozerov [MO15]. It is based on the observation that ISD is a similarity search problem under the Hamming metric. In particular, Eq. (2) defines the approximate relation:

$$\mathbf{Qe}_1 \approx \mathbf{Qe}_2 + \bar{s}. \tag{12}$$

The approximation sign \approx means that the Hamming distance between the left-hand side and the right-hand side of Eq. (12) is at most $wt(\mathbf{e}_3) = w - p$ (cf. Eq. (4)). Enumerating over all \mathbf{e}_1 and \mathbf{e}_2 , we receive an instance of the $(w - p)$ -Near Neighbour (NN) problem:

Definition 3 (γ -Near Neighbour). *Let $\mathcal{L} \subset \mathbb{F}_2^n$ be a list of uniform random binary vectors. The γ -Near Neighbour problem consists in preprocessing \mathcal{L} s.t. upon receiving a query vector $\mathbf{q} \in \mathbb{F}_2^n$, we can efficiently find all $\mathbf{v} \in \mathcal{L}$ that are γ -close to \mathbf{q} , i.e., all \mathbf{v} with $dt(\mathbf{v}, \mathbf{q}) \leq \gamma \cdot n$ for some $\gamma \leq 1/2$.²*

Thus the ISD instance given in Eq. (12) becomes a special case of the $(w - p)$ -NN problem with $\mathcal{L} = \{\mathbf{Q}\mathbf{e}_1\}$ for all $\mathbf{e}_1 \in \mathbb{F}_2^{(k+\ell)/2} \times \mathbf{0}^{(k+\ell)/2}$, and the queries taken from $\{\mathbf{Q}\mathbf{e}_2 + \mathbf{s}\}$ for all $\mathbf{e}_2 \in \mathbf{0}^{(k+\ell)/2} \times \mathbb{F}_2^{(k+\ell)/2}$. In [MO15], the algorithm is described for this special case, namely, when the number of queries is equal to $|\mathcal{L}|$ and all the queries are explicitly given in advance. So it is not immediately clear how to use their result in quantum setting, where we only operate on the sublists of \mathcal{L} and update them with new vectors during the quantum walk.

In this section, we re-phrase the May-Ozerov in the ‘Update’ and ‘Query’ language. It allows us to use the algorithm in more general settings, e.g., when the number of queries differs from $|\mathcal{L}|$ and when queries \mathbf{q} do not come all at once. This view enables us to adapt their algorithm to quantum-walk framework.

The main ingredient of the May-Ozerov algorithm is what became known as Locality-Sensitive Filtering (LSF), see [BDGL16] for an example of this technique in the context of lattice sieving. In LSF we create a set $\mathcal{C} \subset \mathbb{F}_2^n$ of *filtering* vectors \mathbf{c} which divide the Hamming space into (possibly overlapping) regions. These regions are defined as Hamming balls of radius α centred at \mathbf{c} , where α is an LSF-parameter we can choose. So each filtering vector $\mathbf{c} \in \mathcal{C}$ defines a region $\text{Region}_{\mathbf{c}}$ as the set of all vectors that are α -close to \mathbf{c} , namely, $\text{Region}_{\mathbf{c}} = \{\mathbf{v} \in \mathbb{F}_2^n : dt(\mathbf{v}, \mathbf{c}) \leq \alpha\}$. Drawing an analogy with Locality-Sensitive Hashing, these filtering vectors play role of hash-functions. In LSF, instead of defining a function, we define its pre-image.

The preprocessing for the input list \mathcal{L} consists in creating a large enough set \mathcal{C} of filtering vectors and assigning all $\mathbf{v} \in \mathcal{L}$ to their regions (see the $\text{INSERT}(\mathbf{v})$ function in Algorithm 2 below). This assignment defines the LSF buckets as $\text{Bucket}_{\mathbf{c}} = \text{Region}_{\mathbf{c}} \cap \mathcal{L}$. The LSF data structure \mathcal{D} consists of the union of all the buckets. In the course of quantum walk search, we will also need to remove vectors from \mathcal{D} . For that we have the $\text{REMOVE}(\mathbf{v})$ function which deletes \mathbf{v} from all the buckets $\text{Bucket}_{\mathbf{c}}$ containing \mathbf{v} . Note that for each $\text{Bucket}_{\mathbf{c}}$ both $\text{INSERT}()$ and $\text{REMOVE}()$ can be implemented in time $\tilde{O}(\log |\text{Bucket}_{\mathbf{c}}|)$ if we store the buckets as, for example, binary trees. Finally, in order to answer a query \mathbf{q} , we look at all buckets $\text{Bucket}_{\mathbf{c}}$ that are β -close to \mathbf{q} (i.e., all $\mathbf{c} \in \mathcal{C}$ with $dt(\mathbf{q}, \mathbf{c}) \leq \beta$), and we check if any of the vectors stored in these β -close buckets gives a solution

² The (dimensionless) distances we consider here, denoted further γ, α, β , are all $\leq 1/2$, since we can flip the bits of the query point and search for ‘close’ rather than ‘far apart’ vectors.

Algorithm 2. Algorithms for Locality-Sensitive Filtering

Parameters:

- α - the insertion parameter
- β - the query parameter
- γ - the target distance
- \mathcal{C} - the set of filtering vectors
- \mathcal{D} - the LSF data structure: $\mathcal{D} = \cup_{\mathbf{c} \in \mathcal{C}} \text{Bucket}_{\mathbf{c}}$

```

1: function INSERT( $\mathbf{x}$ )                                ▷ Add  $\mathbf{x}$  to all the relevant buckets of  $\mathcal{D}$ 
2:   for all  $\mathbf{c} \in \mathcal{C}$  s.t.  $dt(\mathbf{c}, \mathbf{x}) \leq \alpha$  do
3:      $\text{Bucket}_{\mathbf{c}} \leftarrow \text{Bucket}_{\mathbf{c}} \cup \{\mathbf{x}\}$ 
4:   end for
5: end function

1: function REMOVE( $\mathbf{x}$ )                                ▷ Remove  $\mathbf{x}$  from all buckets
2:   for all  $\mathbf{c} \in \mathcal{C}$  s.t.  $dt(\mathbf{c}, \mathbf{x}) \leq \alpha$  do
3:      $\text{Bucket}_{\mathbf{c}} \leftarrow \text{Bucket}_{\mathbf{c}} \setminus \{\mathbf{x}\}$ 
4:   end for
5: end function

1: function QUERY( $\mathbf{q}$ )                                  ▷ Find all  $\mathbf{x} \in \mathcal{D}$  with  $dt(\mathbf{x}, \mathbf{q}) \leq \beta$ 
2:    $\text{CloseVectors} \leftarrow \emptyset$ 
3:   for all  $\mathbf{c} \in \mathcal{C}$  s.t.  $dt(\mathbf{c}, \mathbf{q}) \leq \beta$  do
4:     for all  $\mathbf{x} \in \text{Bucket}_{\mathbf{c}}$  do
5:       if  $dt(\mathbf{x}, \mathbf{q}) \leq \gamma$  then
6:          $\text{CloseVectors} \leftarrow \text{CloseVectors} \cup \{\mathbf{x}\}$ 
7:       end if
8:     end for
9:   end for
10:  return  $\text{CloseVectors}$ 
11: end function

```

to γ -Near Neighbour. As it is typically the case for NN-algorithms [Laa15], we have two trade-off parameters (α, β): the closer α to 1/2, the more buckets we should create, but the query is fast because we may allow small β . Making α smaller reduces the preprocessing cost but requires more work during queries.

Structured filter-vectors or the ‘strips technique’. In the main procedures of LSF, UPDATE, REMOVE, and QUERY, we are required to find all close buckets for a given point. Naive search finds these buckets time $|\mathcal{C}|$ which is inefficient. We can do better by making filter-vectors \mathbf{c} structured. The technique has several names, ‘strips’ in [MO15], ‘random product code’ in [BDGL16], and ‘tensoring’ in [Chr17], but either way it amounts to the following. Each vector \mathbf{c} is a concatenation of several codewords from some low-dimensional codes (so, \mathcal{C} is a Cartesian product of all these codes). All \mathbf{c} ’s close to \mathbf{x} are obtained by iteratively decoding the relevant projections of \mathbf{x} under the codes defined on these projections (say, for $\mathbf{x} = [\mathbf{x}_1 \parallel \dots \parallel \mathbf{x}_\ell]$, we start by decoding \mathbf{x}_1). On each iteration, we filter out those \mathbf{c} ’s that are guaranteed to be far from \mathbf{x} (i.e, only

$\mathbf{c} = [\mathbf{c}_1 || \dots || \mathbf{c}_\ell]$'s with \mathbf{c}_1 close to \mathbf{x}_1 are kept). Choosing the lengths of low-dimensional codes carefully enough, we can ensure that \mathbf{c} 's are sufficiently close to independent random vectors. This trick allows us find all close buckets in time (up to lower-order terms) equal to the output size. We refer the reader to [BDGL16, MO15] for details.

Before we give complexities for the routines described in Algorithm 2 as functions of α, β, γ , we recall the definition of the entropy function for a discrete probability distribution defined by a vector \mathbf{p} .

Definition 4. Let $\mathbf{p} \in \mathbb{R}^t$ be a real vector that represents a certain probability distribution, i.e., \mathbf{p} satisfies $0 \leq \mathbf{p}_i \leq 1$, and $\sum_{i=1}^t \mathbf{p}_i = 1$. Then $H(\mathbf{p})$ is the entropy of the distribution \mathbf{p} :

$$H(\mathbf{p}) = - \sum_i \mathbf{p}_i \log \mathbf{p}_i.$$

We will be using the above definition in the following context: Let $(\mathbf{x}_1, \dots, \mathbf{x}_m)$ be an m -tuple of vectors from \mathbb{F}_2^n and let $\mathbf{p} \in \mathbb{R}^{2^m}$ be a real vector indexed by all m -length binary vectors that represents the distribution of the m -tuple. That is, $\mathbf{p}_{i_1 \dots i_m}$ counts the number of occurrences (relative to n) of the coordinates' configuration: $\mathbf{p}_{i_1 \dots i_m} = |\{c : \mathbf{x}_1[c] = i_1, \dots, \mathbf{x}_m[c] = i_m\}|$. Such \mathbf{p} defines a discrete probability distribution on $\{1, \dots, 2^m\}$.

For example, consider a random 2-tuple $(\mathbf{x}_1, \mathbf{x}_2)$ with $dt(\mathbf{x}_1, \mathbf{x}_2) = w$. Its distribution vector is $\mathbf{p} = (p_{00}, p_{01}, p_{10}, p_{11})$ satisfying $p_{01} + p_{10} = w$ and $p_{00} = |\{c : \mathbf{x}_1[c] = 0, \mathbf{x}_2[c] = 0\}|$, $p_{01} = |\{c : \mathbf{x}_1[c] = 0, \mathbf{x}_2[c] = 1\}|$, $p_{10} = |\{c : \mathbf{x}_1[c] = 1, \mathbf{x}_2[c] = 0\}|$, $p_{11} = |\{c : \mathbf{x}_1[c] = 1, \mathbf{x}_2[c] = 1\}|$. In case \mathbf{x}_1 is fixed, we can shift the tuple: $(\mathbf{0}, \mathbf{x}_2 - \mathbf{x}_1)$, and obtain $\mathbf{p} = (1 - w, w)$ with $H(\mathbf{p}) = H(w) = -w \log w - (1 - w) \log(1 - w)$, which just counts the number of all binary vectors of weight w .

In the following, we give complexities of the Near Neighbour problem routines. We assume throughout that the target distance $0 \leq \gamma \leq 1/2$.

Lemma 1 (Size of \mathcal{C}). To answer a Near Neighbour query \mathbf{q} with the QUERY(\mathbf{q}) procedure from Algorithm 2, i.e., output all $\mathbf{v} \in \mathcal{L}$ s.t. $dt(\mathbf{q}, \mathbf{v}) \leq \gamma$ with super-exponentially small error³, the total number of buckets \mathcal{C} should be (up to sub-exponential factors)

$$|\mathcal{C}| = 2^{(1 - (H(\mathbf{p}(\alpha, \beta, \gamma)) - H(\mathbf{p}(\gamma))) \cdot n),} \tag{13}$$

where $\mathbf{p}(\alpha, \beta, \gamma) \in \mathbb{R}^8$ and $\mathbf{p}(\gamma) \in \mathbb{R}^4$ are probability distributions that satisfy

$$\mathbf{P}(\alpha, \beta, \gamma) : \begin{cases} p_{000} = p_{111} = \frac{1}{2} - \frac{1}{4}(\gamma + \beta + \alpha) \\ p_{001} = p_{110} = \frac{1}{4}(\gamma + \beta - \alpha) \\ p_{010} = p_{101} = \frac{1}{4}(\gamma + \alpha - \beta) \\ p_{100} = p_{011} = \frac{1}{4}(\beta + \alpha - \gamma) \end{cases} \quad \mathbf{P}(\gamma) : \begin{cases} p_{00} = p_{11} = \frac{1-\gamma}{2} \\ p_{10} = p_{01} = \frac{\gamma}{2}. \end{cases}$$

³ By 'error' we mean missing a vector which is γ -close to \mathbf{q} .

Proof. Consider a pair (\mathbf{v}, \mathbf{q}) s.t. $dt(\mathbf{v}, \mathbf{q}) = \gamma$. The number of filtering vectors $|\mathcal{C}|$ is determined by the inverse of the probability that a random vector \mathbf{c} will ‘find’ this pair, namely

$$|\mathcal{C}| = 1 / \Pr_{\mathbf{c} \in \{0,1\}^n} [dt(\mathbf{c}, \mathbf{v}) = \alpha, dt(\mathbf{c}, \mathbf{q}) = \beta \mid dt(\mathbf{v}, \mathbf{q}) = \gamma]. \tag{14}$$

Here we give a brief explanation why we can do it and provide an extended argument in the full version ([Kir18]). The change is legitimate since we can rewrite the above probability as the sum over all α', β' ranging from 0 to α resp. β . The sum attains its maximum at α, β (otherwise we could have decreased α and/or β keeping the same success probability, but improving the runtime). We argue on the validity sign change for γ via analysis: the probability given in the denominator of Eq. (14) is monotonously increasing as γ decreases.

If we want to find all but super-exponentially small fraction of \mathbf{v} 's for a given \mathbf{q} , we increase $|\mathcal{C}|$ by a poly(n) factor for some large enough polynomial and obtain the result by Chernoff bounds.

The denominator of Eq. (14) is (assuming \mathbf{c}, \mathbf{q} , and \mathbf{v} are uniform)

$$\frac{\Pr_{\mathbf{c}, \mathbf{v}, \mathbf{q}} [dt(\mathbf{c}, \mathbf{v}) = \alpha, dt(\mathbf{c}, \mathbf{q}) = \beta, dt(\mathbf{v}, \mathbf{q}) = \gamma]}{\Pr_{\mathbf{v}, \mathbf{q}} [dt(\mathbf{v}, \mathbf{q}) = \gamma]} = \frac{2^{H(\mathbf{p}(\alpha, \beta, \gamma)) \cdot n} / 2^{3n}}{2^{H(\mathbf{p}(\gamma)) \cdot n} / 2^{2n}} = \frac{1}{|\mathcal{C}|}, \tag{15}$$

for some distribution vectors $\mathbf{p}(\alpha, \beta, \gamma), \mathbf{p}(\gamma)$.

The statement about the entries of the vector $\mathbf{p}(\alpha, \beta, \gamma)$ comes from the following three facts (the entries for $\mathbf{p}(\gamma)$ are straightforward to obtain):

- the distance constraints: three for $\mathbf{p}(\alpha, \beta, \gamma)$ and one for $\mathbf{p}(\gamma)$,
- the uniformity of \mathbf{v}, \mathbf{c} and \mathbf{q} (this allows to assume that the contribution of $p_{01} = p_{010} + p_{011}$ and $p_{10} = p_{100} + p_{101}$ to the distance between two uniform vectors is the same),
- the fact that $\sum_i \mathbf{p}_i = 1$.

This gives us 7 equations for 8 variables leaving 1 degree of freedom. We further assume that $p_{000} = p_{111}$ (essentially, the same choice was done in [MO15, Lemma 2]). Solving these linear equations gives \mathbf{p}_i 's as stated in the theorem. \square

Theorem 2 (LSF complexity for Hamming metric). *For the Near Neighbour problem with some fixed target $0 \leq \gamma \leq 1/2$, the routines given in Algorithm 2 for some fixed $0 \leq \alpha, \beta \leq 1/2$ and the data structure $\mathcal{D} = \cup_{\mathbf{c}} \text{Bucket}_{\mathbf{c}}$, have the following expected costs (up to terms sub-exponential in n):*

- Each UPDATE costs $T_{\text{Upd}}^{\text{LSF}} = |\mathcal{C}| \cdot 2^{(H(\alpha)-1)n}$.
- Preprocessing costs $T_{\text{Prep}}^{\text{LSF}} = |\mathcal{L}| \cdot |\mathcal{C}| \cdot 2^{(H(\alpha)-1)n}$.
- Each QUERY costs $T_{\text{Query}}^{\text{LSF}} = |\mathcal{C}| \cdot 2^{(H(\beta)-1)n} \cdot \mathbb{E}|\text{Bucket}_{\mathbf{c}}|$, where $\mathbb{E}|\text{Bucket}_{\mathbf{c}}|$
- the expected size of each bucket – is equal to $|\mathcal{L}| \cdot 2^{(H(\alpha)-1)n}$.

Proof. We assume that our buckets \mathcal{C} are constructed using ‘structured’ filter vectors \mathbf{c} , which enables us to find all the buckets within a certain distance to a fixed point in the output time (see the discussion above) and, at the same time, allows us to treat \mathbf{c} as (sufficiently) uniform random vectors.

The expected number of buckets for an update parameter α and a fixed \mathbf{v} is $|\mathcal{C}| \cdot \Pr_{\mathbf{c} \in \{0,1\}^n} [dt(\mathbf{v}, \mathbf{c}) = \alpha] = |\mathcal{C}| \cdot \Pr [wt(\mathbf{c} - \mathbf{v}) = \alpha] = |\mathcal{C}| \cdot 2^{H(\mathbf{p}(\alpha))n} = |\mathcal{C}| \cdot 2^{(H(\alpha)-1)n}$. Preprocessing calls $\text{UPDATE}(\mathbf{v})$ for all $\mathbf{v} \in \mathcal{L}$, hence its complexity is $|\mathcal{L}| \cdot T_{\text{upd}}$.

The probability that \mathbf{v} will be added to a certain bucket during the update is again $2^{(H(\alpha)-1)n}$, so after $|\mathcal{L}|$ calls to UPDATE , the average bucket-load will be (up to sub-exponential terms) $|\mathcal{L}| \cdot 2^{(H(\alpha)-1)n}$. Treating $|\text{Bucket}_{\mathbf{c}}|$ as a random variable and using standard Chernoff bound arguments, one can easily show that $|\text{Bucket}_{\mathbf{c}}|$ does not significantly deviate from its expected value.

During the $\text{QUERY}(\mathbf{q})$ calls, we find all β -close buckets in time $|\mathcal{C}| \cdot 2^{(H(\beta)-1)n}$ and for each bucket we look through $|\text{Bucket}_{\mathbf{c}}|$ vectors and among them choose all γ -close to \mathbf{q} . \square

In the application to ISD, where the number of queries is equal to $|\mathcal{L}|$, it makes sense to setup the NN-parameters α and β s.t. the time spent on preprocessing and the time spent on $|\mathcal{L}|$ queries are equal. Indeed, in May-Ozerov algorithm, we have $\alpha = \beta$ and, furthermore, $\alpha = H^{-1}(1 - \log |\mathcal{L}|)$ to make the expected size of buckets equal to 1. After almost trivial algebraic manipulations with Eq. (13) for these parameters, we obtain $\log |\mathcal{C}| = (1 - \gamma) \left(1 - H \left(\frac{H^{-1}(1 - \log |\mathcal{L}|) - \gamma/2}{1 - \gamma} \right) \right)$, which matches the result of [MO15, Theorem 1].

4.2 Quantum ISD with Near Neighbour

Here we explain how to embed the Near Neighbour routines into quantum walk search. In classical setting, we would create two lists $\mathcal{L}_1, \mathcal{L}_2$ of equal size (see Fig. 1), setup the data structure \mathcal{D} (i.e., choose enough filter-vectors) and call $\text{UPDATE}(\mathbf{v}_1)$ for all $\mathbf{v}_1 \in \mathcal{L}_1$ with some update parameter α . This is the preprocessing stage. Then, for each $\mathbf{v}_2 \in \mathcal{L}_2$, we call $\text{QUERY}(\mathbf{v}_2)$ for a query parameter β and search through the output of size $|\mathcal{C}| \cdot 2^{(H(\beta)-1)n} \cdot \mathbb{E}|\text{Bucket}_{\mathbf{c}}|$ for $\mathbf{v}_1 \in \mathcal{D}$ s.t. $dt(\mathbf{v}_1, \mathbf{v}_2) = w - p$. This is the query stage. From the solution pair $(\mathbf{v}_1, \mathbf{v}_2)$, we retrieve the error-vector and solve the 2-list matching problem. If we set $\alpha = \beta$ to balance out the costs for updates and queries, and $\alpha = H^{-1}(1 - \log |\mathcal{L}_1|)$ to balance preprocessing and query stages, we solve the 2-list matching problem for ISD in time $|\mathcal{C}|$ which is exactly what May-Ozerov algorithm achieves.

It is not hard to see that the ‘Update-and-Query’ description of the Near Neighbour search suits particularly well the quantum walk search framework. Assume we run the walk over a superposition of $(S_1, S_2) \subset \mathcal{L}_1 \times \mathcal{L}_2$, where $|S_1 \cup S_2| = \Theta(r)$ for some r which will be determined later. During the Setup phase we create the LSF data structure for S_1 , and call $\text{UPDATE}(\mathbf{v}_1)$ for all $\mathbf{v}_1 \in S_1$. Now, contrary to the classical setting, we apply Grover’s search over all $\mathbf{v}_2 \in S_2$ with the Grover checking function being $\text{QUERY}(\mathbf{v}_2)$, which tells us

Algorithm 3. A quantum walk with Near Neighbour

Quantum walk SETUP:

- 1: Create the LSF data structure \mathcal{D} on the auxiliary register $|Aux\rangle$
- 2: **for all** $\mathbf{v}_1 \in S_1$ **do** $\triangleright |S_1| = |S_2| = \Theta(r)$
- 3: Call UPDATE(\mathbf{v}_1) \triangleright Update \mathcal{D}
- 4: **end for**
- 5: Using Grover search over all $\mathbf{v}_2 \in S_2$: QUERY(\mathbf{v}_2) to check if (S_1, S_2) is marked

Quantum walk UPDATE:

- 6: $S_{\text{new}} \leftarrow \emptyset$
- 7: Repeat $\Theta(\sqrt{|S_1|})$ times:
- 8: Call UPDATE(\mathbf{v}^*) \triangleright Add a new $\mathbf{v}^* \notin S_1 \cup S_2$ to \mathcal{D}
- 9: $S_{\text{new}} \leftarrow S_{\text{new}} \cup \{\mathbf{v}^*\}$
- 10: Call REMOVE(\mathbf{v}) \triangleright Remove $\mathbf{v} \in S_1 \cup S_2$ from \mathcal{D}
- 11: $(S_1, S_2) \leftarrow (S_1, S_2) \setminus \{\mathbf{v}\}$
- 12: Update the register (S_1, S_2) with S_{new}

Quantum walk CHECK:

- 13: Run Grover search over all $\mathbf{v}_2 \in S_{\text{new}}$ using QUERY(\mathbf{v}_2) to check if (S_1, S_2) is marked
-

whether (S_1, S_2) is ‘marked’, i.e., whether it contains $(\mathbf{v}_1, \mathbf{v}_2)$ s.t. $dt(\mathbf{v}_1, \mathbf{v}_2) = w - p$. This allows us to spend more time on QUERY calls choosing $\beta \neq \alpha$.

We do the same during the Update+Check phases: we update $S_1 \cup S_2$ with $\Theta(\sqrt{r})$ new vectors and for each of them we call the LSF UPDATE routine. We also delete $\Theta(\sqrt{r})$ vectors by calling the LSF REMOVE function. The Checking phase of the walk calls QUERY() in the superposition over all $\Theta(\sqrt{r})$ new vectors and decides in time $\Theta(r^{1/4} \cdot T_{\text{Query}}^{\text{LSF}})$ whether the updated $S_1 \cup S_2$ is marked.

So the advantage of quantum walk is two-fold: first, we work only with exponentially shorter sublists S_1, S_2 and, second, during the Checking phase we use Grover over many QUERY calls. Algorithm 3 below summarizes the above description and should be used at Step (2.b) of Algorithm 1.

Finally, one can combine quantum Near Neighbour search with the Shamir-Schroepfel idea: instead of working with long lists $\mathcal{L}_1, \mathcal{L}_2$, consider their sublists $\mathcal{L}'_1, \mathcal{L}'_2 \subset \mathcal{L}_1, \mathcal{L}_2$ s.t. $\mathbf{v}_1 \in \mathcal{L}'_1$ and $\mathbf{v}_2 \in \mathcal{L}'_2$ are equal to a certain vector $\mathbf{t} \in \mathbb{F}_2^{\ell'}$ on ℓ' -coordinates. The probability that $\mathcal{L}'_1, \mathcal{L}'_2$ contain the solution is $2^{-\ell'}$. Quantumly, the cost to construct $\mathcal{L}'_1, \mathcal{L}'_2$ that contain the solution is $2^{-\ell'/2}$ (Grover’s search). Now run NN-search on shorter lists $\mathcal{L}'_1, \mathcal{L}'_2$ and on the dimension reduced by ℓ' . Such algorithm offers a slight improvement both in time and memory over plain Stern’s algorithm as the next theorem shows.

Theorem 3 (Quantum Dumer+Near Neighbour). *Assume we run Algorithm 1 for Dumer’s decoding, where during quantum walk we use the $(w - p)$ -Near Neighbour routines from Algorithm 3. Then the expected running time of Dumer’s algorithm is $\tilde{O}(2^{0.059922 \cdot n + o(n)})$ with quantum memory complexity $\tilde{O}(2^{0.00897 \cdot n + o(n)})$. Using additionally the Shamir-Schroepfel trick, time and memory can be improved to $\tilde{O}(2^{0.059450 \cdot n + o(n)})$ and $\tilde{O}(2^{0.00808 \cdot n + o(n)})$.*

Proof. The number of trials $\mathcal{P} = \mathcal{P}(p, \ell)$ until we find a good permutation π for the Near Neighbour version of Dumer’s decoding is given in Eq. (5). Grover’s search will find a good π in time $\mathcal{O}(\sqrt{\mathcal{P}})$. The checking routine for this search is a quantum walk over the subsets $(S_1, S_2) \subset \mathcal{L}_1 \times \mathcal{L}_2$ with $|\mathcal{L}_1| = |\mathcal{L}_2| = \binom{(k+\ell)/2}{p/2}$, where during the walk we look for an approximate match in $S_1 \cup S_2$ using Algorithm 3. Assume $|S_1 \cup S_2| = r$. We want to determine r and the LSF parameters α, β for UPDATE and QUERY that minimize the Near Neighbour search. In the following we omit the $\tilde{\mathcal{O}}$ -notation for all runtimes.

The complexity of the quantum walk Setup is $\max\{|\mathcal{C}|, r \cdot T_{\text{Upd}}^{\text{LSF}}, \sqrt{r} \cdot T_{\text{Query}}^{\text{LSF}}\}$, where $|\mathcal{C}|$ is given in Eq. (13) and T_{*}^{LSF} is given in Theorem 2. That is, we take the maximum between the time to setup \mathcal{D} , call the UPDATE r times and run Grover over the r new elements to decide on marked subsets for the starting superposition. The decision is realized via calling QUERY.

In the Update phase, we call \sqrt{r} times UPDATE and REMOVE LSF routines to update \mathcal{D} . The complexity of the Update phase is $\sqrt{r} \cdot T_{\text{Upd}}^{\text{LSF}}$, and of the checking phase is $r^{1/4} \cdot T_{\text{Query}}^{\text{LSF}}$. As in the classical case, we set

$$\alpha = H^{-1}(1 - \log r)$$

to guarantee that the expected size of each bucket is 1.⁴ Note that this choice also balances $|\mathcal{C}| = r \cdot T_{\text{Upd}}^{\text{LSF}}$ for the quantum walk Setup. Finally, the quantum walk Checking routine runs Grover search over \sqrt{r} new elements in $S_1 \cup S_2$ to update the ‘marking’ flag for $S_1 \cup S_2$. To balance the Update and the Check phases (i.e, when $\sqrt{r} \cdot T_{\text{Upd}}^{\text{LSF}} = r^{1/4} \cdot T_{\text{Query}}^{\text{LSF}}$), we set

$$\beta = H^{-1}(1 - \frac{3}{4} \log r).$$

Such choice also guarantees that during the Setup, $r \cdot T_{\text{Upd}}^{\text{LSF}} \geq \sqrt{r} \cdot T_{\text{Query}}^{\text{LSF}}$. Moreover, it enables us to setup β slightly larger than α since QUERY becomes cheaper.

Finally, we want to balance T_{S} for Setup, which is $r \cdot T_{\text{Upd}}^{\text{LSF}}$, with the Update and Check phases, $\frac{1}{\sqrt{\varepsilon}} (\frac{1}{\sqrt{\delta}} \cdot T_{\text{U}} + T_{\text{C}})$, cf. Eq. (10). Due to our choices of α, β , this expression is equal to $\frac{1}{\sqrt{\varepsilon}} \cdot \sqrt{r} T_{\text{Upd}}^{\text{LSF}}$ since $\delta \approx 1/r$.

For $k = 2, \varepsilon = r^2/|\mathcal{L}_1 \cup \mathcal{L}_2|^2$, from where we obtain

$$r = |\mathcal{L}_1 \cup \mathcal{L}_2|^{2/3} \approx \left(\binom{(k+\ell)/2}{p/2} \right)^{2/3}.$$

The last parameter we need to determine in order to give the complexity of decoding is the weight parameter p for which we execute the $(w - p)$ -Near Neighbour search. The brute-force search over p reveals that for $p = 0.0027 \cdot n$, $\alpha = 0.4169 \cdot n, \beta = 0.4280 \cdot n$, we have $|\mathcal{C}| = 2^{0.00897 \cdot n}$. We obtain the figures stated in the theorem by computing the necessary number of permutation for such p and noting that $|\mathcal{C}|$ determines the memory cost.

⁴ One could also run Grover inside each bucket during the Query phase, when the buckets are larger than 1. This, however, does not seem to bring an improvement.

If we construct the lists $\mathcal{L}_1, \mathcal{L}_2$ using the Shamir-Schroepel idea, we start with $k = 4$ list $\mathcal{L}_{i,j}$ each of size $\binom{k+\ell}{p/4}^{1/4}$. We merge them into 2 lists $\mathcal{L}_1, \mathcal{L}_2$ by enforcing the vectors $(\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ having the same value on ℓ' coordinates. Quantumly, we find the correct value for the ISD solution in time $2^{\ell'/2}$. We solve the 4-list matching problem via quantum walk with the optimal choice for $r = |\mathcal{L}_{i,j}|^{4/5}$. Optimization reveals that the choosing $p = 0.043, \ell' = 0.007, \alpha = 0.4330, \beta = 0.4419$, gives the best running time. \square

Why choosing larger k does not help. The more starting lists k an ISD algorithm has, the larger the fraction $1/\varepsilon = |L|^k/r^k$ is for any $r < |L|$. Hence, the running time of approximate search and, consequently, the running time of quantum walk become more expensive. The search for optimal parameters tries to shift the work-load to the Grover search for a good permutation by making p smaller (the smaller p is, the harder it is find a good π but the easier the NN-search). From the above theorem, we have for $k = 2, p = 0.0027$ which is already quite small. An optimization for $k = 4$ (e.g., MMT) chooses $p = 0$ which is Prange's algorithm. This is also the reason why we do not get a quantum speed-up for algorithms proposed in [BM17].

Acknowledgements. The author thanks Alexander May for enlightening discussions and suggestions. This work is supported by ERC Starting Grant ERC-2013-StG-335086-LATTAC.

References

- [Amb04] Ambainis, A.: Quantum walk algorithm for element distinctness. In: FOCS, pp. 210–239 (2004)
- [BCA89] Brouwer, A.E., Cohen, A.M., Neumaier, A.: Distance-Regular Graphs. Springer, Heidelberg (1989). <https://doi.org/10.1007/978-3-642-74341-2>
- [BDGL16] Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: SODA 2016, pp. 10–24 (2016)
- [Ber10] Bernstein, D.J.: Grover vs. McEliece. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 73–80. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12929-2_6
- [BJLM13] Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum algorithms for the subset-sum problem. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 16–33. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_2
- [BJMM12] Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: how $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_31
- [BM17] Both, L., May, A.: Optimizing BJMM with nearest neighbors: full decoding in $2^{2n/21}$ and McEliece security. In: The Tenth International Workshop on Coding and Cryptography (2017)

- [CE05] Childs, A.M., Eisenberg, J.M.: Quantum algorithms for subset finding. *Quantum Inf. Comput.* **5**(7), 593–604 (2005)
- [Chr17] Christiani, T.: A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In: SODA, pp. 31–46 (2017)
- [Dum91] Dumer, I.: On minimum distance decoding of linear codes. In: Proceedings of the 5th Joint Soviet-Swedish International Workshop on Information Theory, pp. 50–52 (1991)
- [IM98] Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 604–613 (1998)
- [Kir18] Kirshanova, E.: Improved quantum information set decoding (2018). <http://perso.ens-lyon.fr/elena.kirshanova/Papers/quantumISD.pdf>
- [KT17] Kachigar, G., Tillich, J.-P.: Quantum information set decoding algorithms. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 69–89. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_5
- [Laa15] Laarhoven, T.: Tradeoffs for nearest neighbors on the sphere. CoRR, abs/1511.07527 (2015)
- [McE78] McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. In: Deep Space Network Progress Report, pp. 114–116 (1978)
- [MMT11] May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_6
- [MNRS11] Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. *SIAM J. Comput.* **40**(1), 142–164 (2011)
- [MO15] May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 203–228. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_9
- [Pra62] Prange, E.: The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory* **6**, 5–9 (1962)
- [SS81] Schroepel, R., Shamir, A.: A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.* **10**, 456–464 (1981)
- [Ste89] Stern, J.: A method for finding codewords of small weight. In: Cohen, G., Wolfmann, J. (eds.) Coding Theory 1988. LNCS, vol. 388, pp. 106–113. Springer, Heidelberg (1989). <https://doi.org/10.1007/BFb0019850>

Author Index

- Baena, John 355
Baldi, Marco 3
Balogh, Marko 467
Barengi, Alessandro 3
Barreto, Paulo S. L. M. 248
Bernstein, Daniel J. 487
Bert, Pauline 271
Both, Leif 25
- Cabarcas, Daniel 355
Castelнови, Laurent 165
Chiaraluca, Franco 3
Czajkowski, Jan 185
- de Boer, Koen 101
de Wolf, Ronald 101
Derler, David 419
Ding, Jintai 375
Doliskani, Javad 248
Ducas, Léo 101
- Eaton, Edward 47, 467
El Bansarkhani, Rachid 441
Escudero, Daniel E. 355
- Fouque, Pierre-Alain 271
- Groot Bruinderink, Leon 185
- Hülsing, Andreas 185
- Ikematsu, Yasuhiko 396
- Jeffery, Stacey 101
- Khathuria, Karan 355
Kirshanova, Elena 507
Kölbl, Stefan 205
- Laarhoven, Thijs 292
Lequesne, Matthieu 47
- Mariano, Artur 292
Martinelli, Ange 165
May, Alexander 25
- Misoczki, Rafael 441
Montgomery, Hart 312
- Niederhagen, Ruben 77, 121
Ning, Kai-Chun 121
- Parent, Alex 47
Pelosi, Gerardo 3
Pereira, Geovandro C. C. F. 248
Perlner, Ray 375, 396
Petzoldt, Albrecht 375
Prest, Thomas 165
- Ramacher, Sebastian 419
Renes, Joost 229
Roux-Langlois, Adeline 271
- Sabt, Mohamed 271
Santini, Paolo 3
Sato, Shingo 331
Schaffner, Christian 185
Sendrier, Nicolas 47
Shikata, Junji 331
Simplicio Jr, Marcos A. 248
Slamanig, Daniel 419
Smith-Tone, Daniel 375, 396
Song, Fang 467
Szefer, Jakub 77
- Takagi, Tsuyoshi 396
- Unruh, Dominique 185
- Vates, Jeremy 396
Verbel, Javier 355
- Wang, Wen 77
- Xagawa, Keita 142
- Yang, Bo-Yin 121, 487
Zanon, Gustavo H. M. 248