

Elisabeth Oswald
Marc Fischlin (Eds.)

LNCS 9056

Advances in Cryptology – EUROCRYPT 2015

34th Annual International Conference
on the Theory and Applications of Cryptographic Techniques
Sofia, Bulgaria, April 26–30, 2015, Proceedings, Part I

1
Part I



Springer

Lecture Notes in Computer Science

9056

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 Zürich, Zürich, Switzerland

John C. Mitchell

Stanford University, Stanford, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Dortmund, Germany

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbrücken, Germany

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

Elisabeth Oswald · Marc Fischlin (Eds.)

Advances in Cryptology – EUROCRYPT 2015

34th Annual International Conference on the Theory
and Applications of Cryptographic Techniques
Sofia, Bulgaria, April 26–30, 2015
Proceedings, Part I

Editors

Elisabeth Oswald
University of Bristol
Bristol
UK

Marc Fischlin
Technische Universität Darmstadt
Darmstadt
Germany

ISSN 0302-9743

ISSN 1611-3349 (electronic)

Lecture Notes in Computer Science

ISBN 978-3-662-46799-2
DOI 10.1007/978-3-662-46800-5

ISBN 978-3-662-46800-5 (eBook)

Library of Congress Control Number: 2015935614

LNCS Sublibrary: SL4 – Security and Cryptology

Springer Heidelberg New York Dordrecht London

© International Association for Cryptologic Research 2015

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.

Printed on acid-free paper

Springer-Verlag GmbH Berlin Heidelberg is part of Springer Science+Business Media
(www.springer.com)

Preface

Eurocrypt 2015, the 34th annual International Conference on the Theory and Applications of Cryptographic Techniques, was held during April 26–30, 2015, in Sofia, Bulgaria, and sponsored by the International Association for Cryptologic Research (IACR). Responsible for the local organization were Svetla Nikova, from Katholieke Universiteit Leuven, and Dimitar Jetchev, from EPFL. They were supported by a Local Organizing Committee consisting of Tsonka Baicheva (Institute of Mathematics and Informatics, BAS), Violeta Dacheva (SANS), and Georgi Sharkov (ESI Center Eastern Europe). We are indebted to them for their support.

To accommodate the request by IACR to showcase as many high-quality submissions as possible, the program was organized in two tracks. These tracks ran in parallel with the exception of invited talks, the single best paper, and two papers with honorable mention. Following a popular convention in contemporary cryptography, one track was labeled \mathcal{R} and featured results more closely related to ‘real’ world cryptography, whereas the second track was labeled \mathcal{I} and featured results in a more abstract or ‘ideal’ world.

A total of 194 submissions were considered during the review process, many were of high quality. As usual, all reviews were conducted double-blind and we excluded Program Committee members from discussing submissions for which they had a possible conflict of interest. To account for a desire (by authors and the wider community alike) to maintain the high standard of publications, we allowed for longer submissions such that essential elements of proofs or other form of evidence could be included in the body of the submissions (appendices were not scrutinized by reviewers). Furthermore, a more focused review process was used that consisted of two rounds. In the first round of reviews we solicited three independent reviews per submission. After a short discussion phase among the 38 Program Committee members, just over half of the submissions were retained for the second round. Authors of these retained papers were given the opportunity to comment on the reviews so far. After extensive deliberations in a second round, we accepted 57 papers. The revised versions of these papers are included in these two volume proceedings, organized topically within their respective track.

The review process would have been impossible without the hard work of the Program Committee members and over 210 external reviewers, whose effort we would like to commend here. It has been an honor to work with everyone. The process was enabled by the Web Submission and Review Software written by Shai Halevi and the server was hosted by IACR. We would like to thank Shai for setting up the service on the server and for helping us whenever needed.

The Program Committee decided to honor one submission with the Best Paper Award this year. This submission was “Cryptanalysis of the Multilinear Map over the Integers” authored by Junghee Cheo, Kyoohyung Han, Changmin Lee, Hansol Ryu, and

Damien Stehlé. The two runners-up to the award, “Robust Authenticated-Encryption: AEZ and the Problem that it Solves” (by Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway) and “On the behaviors of affine equivalent Sboxes regarding differential and linear attacks” (by Anne Canteaut and Joëlle Roué) received Honorable Mentions and hence also invitations for the Journal of Cryptology.

In addition to the contributed talks, we had three invited speakers: Kristin Lauter, Tal Rabin, and Vincent Rijmen. We would like to thank them for accepting our invitation and thank everyone (speakers, session chairs, and rump session chair) for their contribution to the program of Eurocrypt 2015.

April 2015

Elisabeth Oswald
Marc Fischlin

EUROCRYPT 2015

The 34th Annual International Conference on the Theory and
Applications of Cryptographic Techniques, Track \mathcal{R}
Sofia, Bulgaria, April 26–30, 2015

General Chairs

- Svetla Nikova Katholieke Universiteit Leuven, Belgium
Dimitar Jetchev École Polytechnique Fédérale de Lausanne,
Switzerland

Program Co-chairs

Program Committee

- | | |
|---------------------|--|
| Masayuki Abe | NTT, Japan |
| Gilles Barthe | IMDEA, Spain |
| Lejla Batina | Radboud University Nijmegen, The Netherlands |
| Alex Biryukov | University of Luxembourg, Luxembourg |
| Alexandra Boldyreva | Georgia Institute of Technology, USA |
| Jan Camenisch | IBM Research – Zurich, Switzerland |
| Anne Canteaut | Inria, France |
| Liqun Chen | HP Laboratories, UK |
| Chen-Mou Cheng | National Taiwan University, Taiwan |
| Marten van Dijk | University of Connecticut, USA |
| Jens Groth | University College London, UK |
| Tetsu Iwata | Nagoya University, Japan |
| Marc Joye | Technicolor, USA |
| Charanjit Jutla | IBM Research, USA |
| Eike Kiltz | Ruhr-Universität Bochum, Germany |
| Markulf Kohlweiss | Microsoft Research, UK |
| Gregor Leander | Ruhr-Universität Bochum, Germany |
| Benoît Libert | ENS Lyon, France |
| Yehuda Lindell | Bar-Ilan University, Israel |
| Stefan Mangard | Graz University of Technology, Austria |
| Steve Myers | Indiana University, USA |
| Gregory Neven | IBM Research – Zurich, Switzerland |

Kaisa Nyberg	Aalto University, Finland
Kenneth G. Paterson	Royal Holloway, University of London, UK
David Pointcheval	École Normale Supérieure Paris, France
Manoj Prabhakaran	University of Illinois at Urbana–Champaign, USA
Emmanuel Prouff	ANSSI, France
Christian Rechberger	Technical University of Denmark, Denmark
Pankaj Rohatgi	Cryptography Research Inc., USA
Alon Rosen	Herzliya Interdisciplinary Center, Herzliya, Israel
Alessandra Scafuro	University of California, Los Angeles, USA
Christian Schaffner	University of Amsterdam, The Netherlands
Dominique Schröder	Saarland University, Germany
Martijn Stam	University of Bristol, UK
François-Xavier Standaert	Université catholique de Louvain, Belgium
Douglas Stebila	Queensland University of Technology, Australia
Frederik Vercauteren	Katholieke Universiteit Leuven, Belgium
Bogdan Warinschi	University of Bristol, UK

External Reviewers

Divesh Aggarwal	Luís T.A.N. Brandão	Markus Dürmuth
Shweta Agrawal	Billy Bob Brumley	Robert Enderlein
Martin Albrecht	Christina Brzuska	Chun-I Fan
Hiroaki Anada	Claude Carlet	Edvard Fargerholm
Prabhanjan Ananth	Angelo De Caro	Pooya Farshim
Elena Andreeva	Ignacio Cascudo	Feng-Hao Liu
Benny Applebaum	David Cash	Matthieu Finiasz
Srinivasan Arunachalam	Andrea Cerulli	Dario Fiore
Gilad Asharov	Pyrros Chaidos	Rob Fitzpatrick
Nuttapong Attrapadung	Yun-An Chang	Robert Fitzpatrick
Saikrishna Badrinarayanan	Jie Chen	Nils Fleischhacker
Rachid El Bansarkhani	Baudoin Collard	Jean-Pierre Flori
Manuel Barbosa	Geoffroy Couteau	Pierre-Alain Fouque
Lynn Batten	Edouard Cuvelier	Thomas Fuhr
Amos Beimel	Joan Daemen	Eiichiro Fujisaki
Sonia Belaid	Vizár Damian	Benjamin Fuller
Josh Benaloh	Jean-Paul Degabriele	Tommaso Gagliardoni
Florian Bergsma	Patrick Derbez	Steven Galbraith
Sanjay Bhattacharjee	David Derler	Nicolas Gama
Nir Bitansky	Christoph Dobraunig	Praveen Gauravaram
Céline Blondeau	Nico Döttling	Ran Gelles
Andrej Bogdanov	Manu Drijvers	Rosario Gennaro
Niek Bouman	Maria Dubovitskaya	Henri Gilbert
Colin Boyd	Orr Dunkelman	Sergey Gorbunov
Elette Boyle	Francois Dupressoir	Matthew Green
Zvika Brakerski	Stefan Dziembowski	Vincent Grosso

Johann Grozschaedl
Sylvain Guilley
Shai Halevi
Michael Hamburg
Mike Hamburg
Fabrice Ben Hamouda
Christian Hanser
Ryan Henry
Jens Hermans
Javier Herranz
Ryo Hiromasa
Shoichi Hirose
Yan Huang
Yuval Ishai
Cess Jansen
Thomas Johansson
Anthony Journault
Antoine Joux
Ali El Kaafarani
Saqib Kakvi
Akshay Kamath
Bhavana Kanukurthi
Carmen Kempka
Dmitry Khrabrovich
Dakshita Khurana
Susumu Kiyoshima
Stefan Koelbl
Francois Koeune
Vlad Kolesnikov
Anna Krasnova
Stephan Krenn
Po-Chun Kuo
Fabien Laguillaumie
Adeline Langlois
Martin M. Lauridsen
Jooyoung Lee
Anja Lehmann
Tancrède Lepoint
Reynald Lercier
Gaëtan Leurent
Anthony Leverrier
Huijia Lin
Steve Lu
Atul Luykx
Giulio Malavolta
Mark Marson

Dan Martin
Christian Matt
Ueli Maurer
Ingo von Maurich
Matthew McKague
Marcel Medwed
Florian Mendel
Bart Mennink
Arno Mittelbach
Payman Mohassel
Mridul Nandi
María Naya-Plasencia
Phong Nguyen
Ryo Nishimaki
Kobbi Nissim
Adam O'Neill
Wakaha Ogata
Miyako Ohkubo
Olya Ohrimenko
Tatsuaki Okamoto
Jiaxin Pan
Omkant Pandey
Omer Paneth
Saurabh Panjwani
Louiza Papachristodolou
Anat Paskin-Cherniavsky
Rafael Pass
Chris Peikert
Ludovic Perret
Léo Perrin
Thomas Peters
Christophe Petit
Duong Hieu Phan
Krzysztof Pietrzak
Benny Pinkas
Jérôme Plût
Christopher Portmann
Romain Poussier
Ignacio Cascudo Pueyo
Ivan Pustogarov
Bertram Pöttering
Max Rabkin
Carla Rafols
Somindu Ramanna
Jothi Rangasamy
Alfredo Rial

Vincent Rijmen
Ben Riva
Matthieu Rivain
Thomas Roche
Mike Rosulek
Ron Rothblum
Yannis Rouselakis
Arnab Roy
Atri Rudra
Kai Samelin
Palash Sarkar
Benedikt Schmidt
Peter Scholl
Peter Schwabe
Gil Segev
Nicolas Sendrier
Yannick Seurin
Abhi Shelat
Adam Shull
Jamie Sikora
Mark Simkin
Daniel Slamanig
Hadi Soleimany
Juarj Somorovsky
Florian Speelman
Damien Stehlé
John Steinberger
Noah Stephens-Davidowitz
Marc Stevens
Pierre-Yves Strub
Stefano Tessaro
Susan Thomson
Mehdi Tibouchi
Tyge Tiessen
Pei-Yih Ting
Elmar Tischhauser
Mike Tunstall
Dominique Unruh
Vinod Vaikuntanathan
Kerem Varici
Vesselin Velichkov
Muthuramakrishnan
Venkatasubramaniam
Daniele Venturi
Nicolas Veyrat-Charvillon

Ivan Visconti
David Wagner
Hoeteck Wee
Erich Wenger
Cyrille Wielding

David Wu
Keita Xagawa
Bo-Yin Yang
Shang-Yi Yang
Kazuki Yoneyama

Mark Zhandry
Vassilis Zikas

Contents – Part I, Track R

Best Paper

Cryptanalysis of the Multilinear Map over the Integers	3
<i>Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé</i>	

Honorable Mentions

Robust Authenticated-Encryption AEZ and the Problem That It Solves	15
<i>Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway</i>	
On the Behaviors of Affine Equivalent Sboxes Regarding Differential and Linear Attacks	45
<i>Anne Canteaut and Joëlle Roué</i>	

Random Number Generators

A Provable-Security Analysis of Intel’s Secure Key RNG	77
<i>Thomas Shrimpton and R. Seth Terashima</i>	
A Formal Treatment of Backdoored Pseudorandom Generators	101
<i>Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart</i>	

Number Field Sieve

Improving NFS for the Discrete Logarithm Problem in Non-prime Finite Fields	129
<i>Razvan Barbulescu, Pierrick Gaudry, Aurore Guillevic, and François Morain</i>	

The Multiple Number Field Sieve with Conjugation and Generalized Joux-Lercier Methods	156
<i>Cécile Pierrot</i>	

Algorithmic Cryptanalysis

Better Algorithms for LWE and LWR	173
<i>Alexandre Duc, Florian Tramèr, and Serge Vaudenay</i>	

On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes	203
<i>Alexander May and Ilya Ozerov</i>	

Symmetric Cryptanalysis I

Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE	231
<i>Itai Dinur</i>	

A Generic Approach to Invariant Subspace Attacks: Cryptanalysis of Robin, iSCREAM and Zorro	254
<i>Gregor Leander, Brice Minaud, and Sondre Rønjom</i>	

Symmetric Cryptanalysis II

Structural Evaluation by Generalized Integral Property	287
<i>Yosuke Todo</i>	

Cryptanalysis of SP Networks with Partial Non-Linear Layers	315
<i>Achiya Bar-On, Itai Dinur, Orr Dunkelman, Virginie Lallemand, Nathan Keller, and Boaz Tsaban</i>	

Hash Functions

The Sum Can Be Weaker Than Each Part	345
<i>Gaëtan Leurent and Lei Wang</i>	

SPHINCS: Practical Stateless Hash-Based Signatures	368
<i>Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O’Hearn</i>	

Evaluating Implementations

Making Masking Security Proofs Concrete: Or How to Evaluate the Security of Any Leaking Device	401
<i>Alexandre Duc, Sébastien Faust, and François-Xavier Standaert</i>	

Ciphers for MPC and FHE	430
<i>Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner</i>	

Masking

Verified Proofs of Higher-Order Masking	457
<i>Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub</i>	
Inner Product Masking Revisited	486
<i>Josep Balasch, Sebastian Faust, and Benedikt Gierlich</i>	

Fully Homomorphic Encryption I

Fully Homomorphic Encryption over the Integers Revisited	513
<i>Jung Hee Cheon and Damien Stehlé</i>	
(Batch) Fully Homomorphic Encryption over Integers for Non-Binary Message Spaces	537
<i>Koji Nuida and Kaoru Kurosawa</i>	

Related-Key Attacks

KDM-CCA Security from RKA Secure Authenticated Encryption	559
<i>Xianhui Lu, Bao Li, and Dingding Jia</i>	
On the Provable Security of the Iterated Even-Mansour Cipher Against Related-Key and Chosen-Key Attacks	584
<i>Benoît Cogliati and Yannick Seurin</i>	

Fully Homomorphic Encryption II

FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second	617
<i>Léo Ducas and Daniele Micciancio</i>	
Bootstrapping for HElib	641
<i>Shai Halevi and Victor Shoup</i>	

Efficient Two-Party Protocols

More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries	673
<i>Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner</i>	
How to Efficiently Evaluate RAM Programs with Malicious Security	702
<i>Arash Afshar, Zhangxiang Hu, Payman Mohassel, and Mike Rosulek</i>	

Symmetric Cryptanalysis III

Cube Attacks and Cube-Attack-Like Cryptanalysis on the Round-Reduced Keccak Sponge Function	733
<i>Itai Dinur, Paweł Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michał Straus</i>	
Twisted Polynomials and Forgery Attacks on GCM	762
<i>Mohamed Ahmed Abdelraheem, Peter Beelen, Andrey Bogdanov, and Elmar Tischhauser</i>	
 Lattices	
Quadratic Time, Linear Space Algorithms for Gram-Schmidt Orthogonalization and Gaussian Sampling in Structured Lattices	789
<i>Vadim Lyubashevsky and Thomas Prest</i>	
Author Index	817

Contents – Part II, Track I

Signatures

Universal Signature Aggregators.	3
<i>Susan Hohenberger, Venkata Koppula, and Brent Waters</i>	

Fully Structure-Preserving Signatures and Shrinking Commitments.	35
<i>Masayuki Abe, Markulf Kohlweiss, Miyako Ohkubo, and Mehdi Tibouchi</i>	

Zero-Knowledge Proofs

Disjunctions for Hash Proof Systems: New Constructions and Applications.	69
<i>Michel Abdalla, Fabrice Benhamouda, and David Pointcheval</i>	

Quasi-Adaptive NIZK for Linear Subspaces Revisited	101
<i>Eike Kiltz and Hoeteck Wee</i>	

Leakage-Resilient Cryptography

Leakage-Resilient Circuits Revisited – Optimal Number of Computing Components Without Leak-Free Hardware.	131
<i>Dana Dachman-Soled, Feng-Hao Liu, and Hong-Sheng Zhou</i>	

Noisy Leakage Revisited	159
<i>Stefan Dziembowski, Sebastian Faust, and Maciej Skorski</i>	

Garbled Circuits

Privacy-Free Garbled Circuits with Applications to Efficient Zero-Knowledge.	191
<i>Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi</i>	

Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits Using Half Gates	220
<i>Samee Zahur, Mike Rosulek, and David Evans</i>	

Crypto Currencies

- One-Out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin 253
Jens Groth and Markulf Kohlweiss

- The Bitcoin Backbone Protocol: Analysis and Applications 281
Juan Garay, Aggelos Kiayias, and Nikos Leonardos

Secret Sharing

- Linear Secret Sharing Schemes from Error Correcting Codes
and Universal Hash Functions 313
*Ronald Cramer, Ivan Bjerre Damgård, Nico Döttling,
Serge Fehr, and Gabriele Spini*

- Function Secret Sharing 337
Elette Boyle, Niv Gilboa, and Yuval Ishai

Outsourcing Computations

- Cluster Computing in Zero Knowledge 371
Alessandro Chiesa, Eran Tromer, and Madars Virza

- Hosting Services on an Untrusted Cloud 404
Dan Boneh, Divya Gupta, Ilya Mironov, and Amit Sahai

Obfuscation and E-Voting

- How to Obfuscate Programs Directly 439
Joe Zimmerman

- End-to-End Verifiable Elections in the Standard Model 468
Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang

Multi-party Computations

- Cryptographic Agents: Towards a Unified Theory of Computing
on Encrypted Data 501
Shashank Agrawal, Shweta Agrawal, and Manoj Prabhakaran

- Executable Proofs, Input-Size Hiding Secure Computation
and a New Ideal World 532
Melissa Chase, Rafail Ostrovsky, and Ivan Visconti

Encryption

Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation	563
--	-----

*Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai,
Mark Zhandry, and Joe Zimmerman*

Improved Dual System ABE in Prime-Order Groups via Predicate Encodings	595
---	-----

Jie Chen, Romain Gay, and Hoeteck Wee

Resistant Protocols

Resisting Randomness Subversion: Fast Deterministic and Hedged Public-Key Encryption in the Standard Model	627
---	-----

Mihir Bellare and Viet Tung Hoang

Cryptographic Reverse Firewalls	657
---	-----

Ilya Mironov and Noah Stephens-Davidowitz

Key Exchange

Mind the Gap: Modular Machine-Checked Proofs of One-Round Key Exchange Protocols	689
---	-----

*Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech,
and Benedikt Schmidt*

Authenticated Key Exchange from Ideal Lattices	719
--	-----

*Jiang Zhang, Zhenfeng Zhang, Jintai Ding, Michael Snook,
and Özgür Dagdelen*

Quantum Cryptography

Non-Interactive Zero-Knowledge Proofs in the Quantum Random Oracle Model	755
---	-----

Dominique Unruh

Privacy Amplification in the Isolated Qubits Model	785
--	-----

Yi-Kai Liu

Discrete Logarithms

Generic Hardness of the Multiple Discrete Logarithm Problem	817
---	-----

Aaram Yun

Author Index	837
-------------------------------	-----

Best Paper

Cryptanalysis of the Multilinear Map over the Integers

Jung Hee Cheon^{1(✉)}, Kyoohyung Han¹, Changmin Lee¹,
Hansol Ryu¹, and Damien Stehlé²

¹ Seoul National University (SNU), Seoul, Republic of Korea
jhcheon@snu.ac.kr

² ENS de Lyon, Laboratoire LIP (U. Lyon, CNRS, ENSL, INRIA, UCBL),
Lyon, France

Abstract. We describe a polynomial-time cryptanalysis of the (approximate) multilinear map of Coron, Lepoint and Tibouchi (CLT). The attack relies on an adaptation of the so-called *zeroizing* attack against the Garg, Gentry and Halevi (GGH) candidate multilinear map. Zeroizing is much more devastating for CLT than for GGH. In the case of GGH, it allows to break generalizations of the Decision Linear and Subgroup Membership problems from pairing-based cryptography. For CLT, this leads to a total break: all quantities meant to be kept secret can be efficiently and publicly recovered.

Keywords: Multilinear maps · Graded encoding schemes

1 Introduction

Cryptographic bilinear maps, made possible thanks to pairings over elliptic curves, have led to a bounty of exciting cryptographic applications. In 2002, Boneh and Silverberg [BS02] formalized the concept of cryptographic multilinear maps and provided two applications: a one-round key multi-party exchange protocol, and a very efficient broadcast encryption scheme. But these promising applications were only day-dreaming exercises, as no realization of such multilinear maps was known. This was changed about ten years later, as Garg, Gentry and Halevi proposed the first approximation to multilinear maps [GGH13a]. They introduced the concept of (approximate) graded encoding scheme as a variant of multilinear maps, and described a candidate construction relying on ideal lattices (which we will refer to as GGH in this work). Soon after, Coron, Lepoint and Tibouchi [CLT13] proposed another candidate construction of a graded encoding scheme, relying on a variant of the approximate greatest common divisor problem (CLT, for short).

The GGH and CLT constructions share similarities. Both are derived from a homomorphic encryption scheme (Gentry’s scheme [Gen09] and the van Dijk *et al.* scheme [DGHV10], respectively). And both rely on some extra public data, called the zero-testing or extraction parameter, which allows to publicly decide

whether the plaintext data hidden in a given encoding is zero, as long as the encoding is not the output of a too deep homomorphic evaluation circuit.

Graded encoding schemes serve as a basis to define presumably hard problems. These problems are then used as security foundations of cryptographic constructions. A major discrepancy between GGH and CLT is that some natural problems seem easy when instantiated with the GGH graded encoding scheme, and hard for CLT. Two such problems are subgroup membership (SubM) and decision linear (DLIN). Roughly speaking, SubM asks to distinguish between encodings of elements of a group and encodings of elements of a subgroup thereof. DLIN consists in determining whether a matrix of elements is singular, given as input encodings of those elements. Another similar discrepancy seems to exist between the asymmetric variants of GGH and CLT: the External Decision Diffie-Hellman (XDH) problem seems hard for CLT but is easy for GGH. XDH is exactly DDH for one of the components of the asymmetric graded encoding scheme. These problems have been extensively used in the context of cryptographic bilinear maps [Sco02, BBS04, BGN05].

In the first public version of [GGH13a] (dated 29 Oct. 2012),¹ the GGH construction was thought to provide secure DLIN instantiation. It was soon realized that DLIN could be broken in polynomial-time. The attack consists in multiplying an encoding of some element m by an encoding of 0 and by the zero-testing parameter: this produces a small element (because the encoded value is $m \cdot 0 = 0$), which happens to be a multiple of m . This *zeroizing attack* (also called weak discrete logarithm attack) is dramatic for SubM, DLIN and XDH. Fortunately, it does not seem useful against other problems, such as Graded Decision Diffie Hellman (GDDH), the adaptation of DDH to the graded encoding scheme setting. As no such attack was known for CLT, the presumed hardness of the CLT instantiations of SubM, DLIN and XDH was exploited as a security grounding for several cryptographic constructions [ABP14, Att14, BP13, BLMR13, GHZ14a, GHZ14b, GLW14, GLSW14, LMR14, Zha14, Zim14].

Main Result. We describe a zeroizing attack on the CLT graded encoding scheme. It runs in polynomial-time, and allows to publicly compute all the parameters of the CLT scheme that were supposed to be kept secret.

Impact of the Attack. The CLT candidate construction should be considered broken, unless the low-level encodings of 0 are not made public. At the moment, there does not remain any candidate multilinear map approximation for which any of SubM, DLIN and XDH is hard. Several recent cryptographic constructions cannot be realized anymore: this includes all constructions from [Att14, GHZ14a, GHZ14b, Zha14], the GPAKE construction of [ABP14] for more than 3 users, one of the two constructions of password hashing of [BP13], the alternative key-homomorphic PRF construction from [BLMR13], and the use of the latter in [LMR14].

Our attack heavily relies on the fact that low-level encodings of 0 are made publicly available. It is not applicable if these parameters are kept secret. They are used

¹ It can be accessed from the IACR eprint server.

in applications to homomorphically re-randomize encodings, in order to “canonicalize” their distributions. A simple way to thwart the attack is to not make any low-level encoding of 0 public. This approach was used in [GGH+13b] and [BR13], for example. It seems that this approach can be used to secure the construction from [Zim14] as well.

Related Works. A third candidate construction of a variant of graded encoding schemes was recently proposed in [GGH14]. In that scheme, no encoding of 0 is provided, as it would incur serious security issues (see [GGH14, Se. 4]).

Our attack was extended in [BWZ14, GHMS14] to settings in which no low-level encoding of 0 is available. The extensions rely on low-level encodings of elements corresponding to orthogonal vectors, and impact [GLW14, GLSW14].

After our attack was published, the draft [GGHZ14a] was updated, to propose a candidate immunization against our attack (see [GGHZ14a, Se. 6]).² Another candidate immunization was proposed in [BWZ14]. Both immunizations have been showed insecure in [CLT14a].

Open Problems. A natural line of research is to extend the range of applications of graded encoding schemes for which the encodings of zero are not needed.

Publishing encodings of zero as well as a zero-test parameter can lead to damaging consequences (total break of CLT, weakness of SubM, DLIN and XDH for GGH). An impossibility result would be fascinating.

Organization. In Section 2, we recall the CLT scheme and the zeroizing attack against GGH. In Section 3, we present our attack on CLT.

2 Preliminaries

Notation. We use $a \leftarrow A$ to denote the operation of uniformly choosing an element a from a finite set A . We define $[n] = \{1, 2, \dots, n\}$. We let \mathbb{Z}_q denote the ring $\mathbb{Z}/(q\mathbb{Z})$. For pairwise coprime integers p_1, p_2, \dots, p_n , we define $\text{CRT}_{(p_1, p_2, \dots, p_n)}(r_1, r_2, \dots, r_n)$ (abbreviated as $\text{CRT}_{(p_i)}(r_i)$) as the unique integer in $(-\frac{1}{2} \prod_{i=1}^n p_i, \frac{1}{2} \prod_{i=1}^n p_i]$ which is congruent to $r_i \bmod p_i$ for all $i \in [n]$. We use the notation $[t]_p$ for integers t and p to denote the reduction of t modulo p into the interval $(-p/2, p/2]$.

We use lower-case bold letters to denote vectors whereas upper-case bold letters are used to denote matrices. For matrix \mathbf{S} , we denote by \mathbf{S}^T the transpose of \mathbf{S} . We define $\|\mathbf{S}\|_\infty = \max_i \sum_{j \in [n]} |s_{ij}|$, where s_{ij} is the (i, j) component of \mathbf{S} . Finally we denote by $\text{diag}(a_1, \dots, a_n)$ the diagonal matrix with diagonal coefficients equal to a_1, \dots, a_n .

2.1 A Candidate Multilinear Map over the Integers

First, we briefly recall the Coron *et al.* construction. We refer to the original paper [CLT13] for a complete description.

² The former version that was impacted by our attack can still be accessed from the IACR eprint server.

The scheme relies on the following parameters.

- λ : the security parameter
- κ : the multilinearity parameter
- ρ : the bit length of the randomness used for encodings
- α : the bit length of the message slots
- η : the bit length of the secret primes p_i
- n : the number of distinct secret primes
- τ : the number of level-1 encodings of zero in public parameters
- ℓ : the number of level-0 encodings in public parameters
- ν : the bit length of the image of the multilinear map
- β : the bit length of the entries of the zero-test matrix H

Coron *et al.* suggested to set the parameters so that the following conditions are met:

- $\rho = \Omega(\lambda)$: to avoid brute force attack (see also [LS14] for a constant factor improvement).
- $\alpha = \lambda$: so that the ring of messages $\mathbb{Z}_{g_1} \times \dots \times \mathbb{Z}_{g_n}$ does not contain a small subring \mathbb{Z}_{g_i} .³
- $n = \Omega(\eta \cdot \lambda)$: to thwart lattice reduction attacks.
- $\ell \geq n \cdot \alpha + 2\lambda$: to be able to apply the leftover hash lemma from [CLT13, Le. 1].
- $\tau \geq n \cdot (\rho + \log_2(2n)) + 2\lambda$: to apply leftover hash lemma from [CLT13, Se. 4].
- $\beta = \Omega(\lambda)$: to avoid the so-called gcd attack.
- $\eta \geq \rho_\kappa + \alpha + 2\beta + \lambda + 8$, where ρ_κ is the maximum bit size of the random r_i 's a level- κ encoding. When computing the product of κ level-1 encodings and an additional level-0 encoding, one obtains $\rho_\kappa = \kappa \cdot (2\alpha + 2\rho + \lambda + 2\log_2 n + 2) + \rho + \log_2 \ell + 1$.
- $\nu = \eta - \beta - \rho_f - \lambda - 3$: to ensure zero-test correctness.

Instance generation: $(\text{params}, p_{zt}) \leftarrow \text{InstGen}(1^\lambda, 1^\kappa)$. Set the scheme parameters as explained above. For $i \in [n]$, generate η -bit primes p_i , α -bit primes g_i , and compute $x_0 = \prod_{i \in [n]} p_i$. Sample $z \leftarrow \mathbb{Z}_{x_0}$. Let $\Pi = (\pi_{ij}) \in \mathbb{Z}^{n \times n}$ with $\pi_{ij} \leftarrow (n2^\rho, (n+1)2^\rho) \cap \mathbb{Z}$ if $i = j$, otherwise $\pi_{ij} \leftarrow (-2^\rho, 2^\rho) \cap \mathbb{Z}$. For $i \in [n]$, generate $\mathbf{r}_i \in \mathbb{Z}^n$ by choosing randomly and independently in the half-open parallelepiped spanned by the columns of the matrix Π and denote by r_{ij} the j -th component of \mathbf{r}_i . Generate $\mathbf{H} = (h_{ij}) \in \mathbb{Z}^{n \times n}$, $\mathbf{A} = (a_{ij}) \in \mathbb{Z}^{n \times \ell}$ such that \mathbf{H} is invertible

³ In fact, it seems that making the primes g_i public, equal, and $\Omega(\kappa)$ may not lead to any specific attack [CLT14b].

and $\|\mathbf{H}^T\|_\infty \leq 2^\beta$, $\|(\mathbf{H}^{-1})^T\|_\infty \leq 2^\beta$ and for $i \in [n]$, $j \in [\ell]$, $a_{ij} \leftarrow [0, g_i]$. Then define:

$$\begin{aligned} y &= \text{CRT}_{(p_i)} \left(\frac{r_i g_i + 1}{z} \right), \text{ where } r_i \leftarrow (-2^\rho, 2^\rho) \cap \mathbb{Z} \text{ for } i \in [n], \\ x_j &= \text{CRT}_{(p_i)} \left(\frac{r_{ij} g_i}{z} \right) \text{ for } j \in [\tau], \\ x'_j &= \text{CRT}_{(p_i)}(x'_{ij}), \text{ where } x'_{ij} = r'_{ij} g_i + a_{ij} \text{ and } r'_{ij} \leftarrow (-2^\rho, 2^\rho) \cap \mathbb{Z} \text{ for } i \in [n], j \in [\ell], \\ (\mathbf{p}_{zt})_j &= \left[\sum_{i=1}^n [h_{ij} \cdot (z^\kappa \cdot g_i^{-1})]_{p_i} \cdot \prod_{i' \neq i} p_{i'} \right]_{x_0} \text{ for } j \in [n]. \end{aligned}$$

Output $\text{params} = (n, \eta, \alpha, \rho, \beta, \tau, \ell, \nu, y, \{x_j\}, \{x'_j\}, \{\Pi_j\}, s)$ and \mathbf{p}_{zt} . Here s is a seed for a strong randomness extractor, which is used for an ‘‘Extraction’’ procedure. We do not recall the latter as it is not needed to describe our attack.

Re-randomizing level-1 encodings: $c' \leftarrow \text{reRand}(\text{params}, c)$. For $j \in [\tau], i \in [n]$, sample $b_j \leftarrow \{0, 1\}$, $b'_i \leftarrow [0, 2^\mu] \cap \mathbb{Z}$, with $\mu = \rho + \alpha + \lambda$. Return $c' = [c + \sum_{j \in [\tau]} b_j \cdot x_j + \sum_{i \in [n]} b'_i \cdot \Pi_i]_{x_0}$. Note that this is the only procedure in the CLT multilinear map that uses the x_j ’s.⁴

Adding and multiplying encodings: $\text{Add}(c_1, c_2) = [c_1 + c_2]_{x_0}$ and $\text{Mul}(c_1, c_2) = [c_1 \cdot c_2]_{x_0}$.

Zero-testing: $\text{isZero}(\text{params}, \mathbf{p}_{zt}, u_\kappa) = ? 0/1$. Given a level- κ encoding c , return 1 if $\|[\mathbf{p}_{zt} \cdot c]_{x_0}\|_\infty < x_0 \cdot 2^{-\nu}$, and return 0 otherwise.

Coron *et al.* also described a variant where only one such $(\mathbf{p}_{zt})_j$ is given out, rather than n of them (see [CLT13, Se. 6]). Our attack requires only one $(\mathbf{p}_{zt})_j$. In [GLW14, App. B.3], Gentry *et al.* described a variant of the above construction that aims at generalizing asymmetric cryptographic bilinear maps. Our attack can be adapted to that variant.

2.2 Zeroizing Attack on GGH

As a warm-up before describing the zeroizing attack on CLT, we recall the zeroizing attack on GGH.

Garg *et al.* constructed the first approximation to multilinear maps, by using ideal lattices [GGH13a]. They used the polynomial ring $R = \mathbb{Z}[x]/(x^n + 1)$ and a (prime) principal ideal $\mathcal{I} = \langle \mathbf{g} \rangle \subseteq R$, where \mathbf{g} is a secret short element. They also chose an integer parameter q and another random secret $\mathbf{z} \in R_q = R/(qR)$. Then one can encode an element of R/\mathcal{I} , via division by \mathbf{z} in R_q . More precisely, a level- i encoding of the coset $\mathbf{e} + \mathcal{I}$ is an element of the form $[\mathbf{c}/\mathbf{z}^i]_q$, where $\mathbf{c} \in \mathbf{e} + \mathcal{I}$ is short. By publishing a zero-testing parameter, any user can decide whether two elements encode the same coset or not.

⁴ This procedure can be adapted to higher levels $1 < k \leq \kappa$ by publishing appropriate quantities in **params**.

The zero-testing parameter is $\mathbf{p}_{zt} = [\mathbf{h} \cdot \mathbf{z}^\kappa / \mathbf{g}]_q$, where \mathbf{h} is appropriately small. For a given level- κ encoding $\mathbf{u} = [\mathbf{c}/\mathbf{z}^\kappa]_q$, the quantity $[\mathbf{u} \cdot \mathbf{p}_{zt}]_q = [\mathbf{h} \cdot \mathbf{c}/\mathbf{g}]_q$ is small if and only if $\mathbf{c} \in \mathcal{I}$, i.e., \mathbf{u} is an encoding of zero.

The latter creates a weakness in the scheme, which enables to solve the Subgroup Membership (SubM) and the decision linear (DLIN) problems easily, by so-called “zeroizing” attack. It uses the property that an encoding of zero has small value when it is multiplied by the zero-testing parameter. In that case, the reduction modulo q is vacuous, and one can have equations over R (instead of R_q) and compute some fixed multiples of secrets. The attack procedure can be summarized as follows (and refer the reader to [GGH13a] for a more detailed description). It relies on the following public parameters:

- $\mathbf{y} = [\mathbf{a}/\mathbf{z}]_q$, with $\mathbf{a} \in 1 + \mathcal{I}$ and \mathbf{a} small, a level-1 encoding of 1,
- $\mathbf{x}_j = [\mathbf{b}_j \mathbf{g}/\mathbf{z}]_q$, with \mathbf{b}_j small, a level-1 encoding of 0,
- $\mathbf{p}_{zt} = [\mathbf{h} \mathbf{z}^\kappa / \mathbf{g}]_q$, with $\mathbf{h} \in R$ appropriately small, the zero-testing parameter.

Step 1: Compute level- κ encodings of zero and get the equations in R by multiplying by the zero-testing parameter.

Let $\mathbf{u} = \mathbf{d}/\mathbf{z}^t$ be a level- t encoding of some message $\mathbf{d} \bmod \mathcal{I}$. Then compute

$$\begin{aligned} \mathbf{f} &:= [\mathbf{u} \cdot \mathbf{x}_j \cdot \mathbf{p}_{zt} \cdot \mathbf{y}^{\kappa-t-1}]_q = \left[\frac{\mathbf{d}}{\mathbf{z}^t} \cdot \frac{\mathbf{b}_j \cdot \mathbf{g}}{\mathbf{z}} \cdot \frac{\mathbf{h} \cdot \mathbf{z}^\kappa}{\mathbf{g}} \cdot \frac{\mathbf{a}^{\kappa-t-1}}{\mathbf{z}^{\kappa-t-1}} \right]_q \\ &= \underbrace{\mathbf{d} \cdot \mathbf{b}_j \cdot \mathbf{h} \cdot \mathbf{a}^{\kappa-t-1}}_{\ll q}. \end{aligned}$$

Note that the last term in the above equation consists of only small elements, so that the equality holds without modulus reduction by q . Therefore we can obtain various multiples of \mathbf{h} (in R) for various \mathbf{u} and \mathbf{x}_j .

Step 2: From multiples of \mathbf{h} , compute a basis of $\langle \mathbf{h} \rangle$. Using a similar procedure, compute a basis of $\langle \mathbf{h} \cdot \mathbf{g} \rangle$, and hence a basis for \mathcal{I} (by dividing $\langle \mathbf{h} \cdot \mathbf{g} \rangle$ by $\langle \mathbf{h} \rangle$).

SubM is as follows: Given a level-1 encoding $\mathbf{u} = [\mathbf{d}/\mathbf{z}]_q$, assess whether $\mathbf{d} \in \langle \mathbf{g}_1 \rangle$, where $\mathbf{g} = \mathbf{g}_1 \cdot \mathbf{g}_2$ (note that in this context, \mathcal{I} is not a prime ideal). Using the above method, we can get $\mathbf{f} = \mathbf{d} \cdot \Delta$ for some Δ (which is unrelated to \mathbf{g}). Taking the gcd of $\langle \mathbf{f} \rangle$ and \mathcal{I} , we easily solve the subgroup membership problem.

DLIN is as follows: Given level- t encodings $\mathbf{C} = (\mathbf{c}_{ij})_{i,j \in [N]}$ of messages $\mathbf{M} = (\mathbf{m}_{ij})_{i,j \in [N]}$ for some $t < \kappa$ and $N > \kappa/t$,⁵ assess whether the rank of \mathbf{M} (over the field R/\mathcal{I}) is full or not. Using the above, we can compute $\mathbf{M} \cdot \Delta$ for some scalar $\Delta \in R/\mathcal{I}$, which is unlikely to be 0. In that case, the matrices $\mathbf{M} \cdot \Delta$ and \mathbf{M} have equal rank, and the problem is easy to solve.

3 A Zeroizing Attack on CLT

The first step of the attack is similar to that of the zeroizing attack of GGH. We compute many level- κ encodings of zero and multiply them by the zero-testing parameter. Then we get matrix equations over \mathbb{Q} (not reduced modulo x_0). By adapting the latter

⁵ If N is smaller than that, the problem is not interesting as it can always be solved efficiently using the zero-test parameter.

to CLT, one would obtain samples from the ideal $\langle h_1, \dots, h_n \rangle \subseteq \mathbb{Z}$. Most of the time, it is the whole \mathbb{Z} , and the samples do not contain any useful information. Instead, we form matrix equations by using several x_j 's rather than a single one.

These equations share common terms. The second step of the attack is to remove some of these common terms by computing the ratio (over the rationals) between two such equations, and to extract the ratios of the CRT components of the underlying plaintexts by computing the eigenvalues.

The third step consists in recovering the p_i 's from these CRT components. Once the p_i 's are obtained, recovering the other secret quantities is relatively straightforward.

Now we give full details of each step.

3.1 Constructing Matrix Equations over \mathbb{Z}

Let $t \leq \kappa - 1$. Let c be a level- t encoding of $(m_1^{(c)}, \dots, m_n^{(c)})$, i.e., $c = c_i/z^t \pmod{p_i}$ and $c_i = m_i^{(c)}$ for all $i \in [n]$. Then we can compute the following quantities using the public parameters (for $j \in [\ell], k \in [\tau]$):

$$\begin{aligned} w_{jk} := [c \cdot x'_j x_k \cdot y^{\kappa-t-1} \cdot (\mathbf{p}_{zt})_1]_{x_0} &= \left[\sum_{i=1}^n [h_{i1} \cdot c \cdot x'_j x_k y^{\kappa-t-1} z^\kappa g_i^{-1}]_{p_i} \cdot \frac{x_0}{p_i} \right]_{x_0} \\ &= \left[\sum_{i=1}^n h_{i1} c_i x'_{ij} r_{ik} (r_i g_i + 1)^{\kappa-t-1} \cdot \frac{x_0}{p_i} \right]_{x_0} \\ &= \left[\sum_{i=1}^n x'_{ij} h'_i c_i r_{ik} \right]_{x_0}, \end{aligned}$$

where $h'_i := h_{i1} (r_i g_i + 1)^{\kappa-t-1} x_0 / p_i$ for $i \in [n]$.

Now, as c is a level- t encoding, then $x'_j \cdot (c \cdot x_k \cdot y^{\kappa-t-1})$ is a valid level- κ Diffie-Hellman product (i.e., a product of one level-0 encoding and κ level-1 encodings). Further, it is an encoding of 0, as x_k is an encoding of 0. By design, we have that $|w_{jk}|$ is much smaller than x_0 (this may be checked by a tedious computation, but this is exactly how the correctness requirement for the zero-test parameter is derived). As a result, the equation $w_{jk} = \sum_{i \in [n]} x'_{ij} h'_i c_i r_{ik}$ holds over the integers.

This equation can be rewritten as follows:

$$w_{jk} = (x'_{1j}, \dots, x'_{nj}) \cdot \text{diag}(c_1, \dots, c_n) \cdot \text{diag}(h'_1, \dots, h'_n) \cdot (r_{1k}, \dots, r_{nk})^T.$$

By letting the index pair (j, k) vary in $[n] \times [n]$, we obtain a matrix equation involving the following matrix $\mathbf{W}_c = (w_{jk}) \in \mathbb{Z}^{n \times n}$.

$$\begin{aligned} \mathbf{W}_c &= \begin{pmatrix} x'_{11} & \cdots & x'_{n1} \\ & \ddots & \\ x'_{1n} & \cdots & x'_{nn} \end{pmatrix} \begin{pmatrix} c_1 & & 0 \\ & \ddots & \\ 0 & & c_n \end{pmatrix} \begin{pmatrix} h'_1 & & 0 \\ & \ddots & \\ 0 & & h'_n \end{pmatrix} \begin{pmatrix} r_{11} & \cdots & r_{1n} \\ & \ddots & \\ r_{n1} & \cdots & r_{nn} \end{pmatrix} \\ &= \mathbf{X}' \quad \text{diag}(c_1, \dots, c_n) \text{ diag}(h'_1, \dots, h'_n) \quad \mathbf{R}. \end{aligned} \tag{1}$$

To build these equations, we need sufficiently many x'_j 's and x_k 's. Namely, we need $\ell \geq n$ and $\tau \geq n$. The design conditions on ℓ and τ ensure that this is the case.

Note that the only component in the right hand side of Equation (1) that depends on c is $\text{diag}(c_1, \dots, c_n)$: the matrices \mathbf{X}' , \mathbf{R} and $\text{diag}(h'_1, \dots, h'_n)$ are independent of c .

3.2 Breaking into the CRT Decomposition

We now take $t = 0$, and instantiate Equation (1) twice, with $c = x'_1$ and $c = x'_2$. We obtain, for $j \in \{1, 2\}$:

$$\mathbf{W}_j := \mathbf{X}' \cdot \text{diag}(x'_{1j}, \dots, x'_{nj}) \cdot \text{diag}(h'_1, \dots, h'_n) \cdot \mathbf{R}.$$

We can then compute (over \mathbb{Q}):

$$\mathbf{W}_1 \cdot \mathbf{W}_2^{-1} = \mathbf{X}' \cdot \text{diag}\left(\frac{x'_{11}}{x'_{12}}, \dots, \frac{x'_{n1}}{x'_{n2}}\right) \mathbf{X}'^{-1}.$$

In the latter, we need that \mathbf{W}_2 is invertible. Below, we will also need that \mathbf{W}_1 is invertible. We argue here that we may assume this is the case. We prove it for \mathbf{W}_1 . Note first that the x'_{ij} 's and the h'_i 's are all non-zero, with overwhelming probability. Note that by design, the matrix $(r_{ij})_{i \in [n], j \in [\tau]}$ has rank n (see [CLT13, Se. 4]). The same holds for the matrix $(x'_{ij})_{i \in [n], j \in [\ell]}$ (see [CLT13, Le. 1]). As we can compute the rank of a $\mathbf{W}_c \in \mathbb{Z}^{t \times t}$ obtained by using an $\mathbf{X}' \in \mathbb{Z}^{t \times n}$ and an $\mathbf{R} \in \mathbb{Z}^{n \times t}$ obtained by respectively using a t -subset of the x'_j 's and a t -subset of the x_j 's, without loss of generality we may assume that our $\mathbf{X}', \mathbf{R} \in \mathbb{Z}^{n \times n}$ are non-singular. The cost of finding such a pair $(\mathbf{X}', \mathbf{R})$ is bounded as $\tilde{\mathcal{O}}((\tau + \ell) \cdot (n^\omega \log x_0)) = \tilde{\mathcal{O}}(\kappa^{\omega+3} \lambda^{2\omega+6})$, with $\omega \leq 2.38$ (assuming all parameters are set smallest possible so that the bounds of Subsection 2.1 hold). Here we used the fact that the rank of a matrix $\mathbf{A} \in \mathbb{Z}^{n \times n}$ may be computed in time $\tilde{\mathcal{O}}(n^\omega \log \|\mathbf{A}\|_\infty)$ (see [Sto09]). This dominates the overall cost of the attack.

As \mathbf{X}' is non-singular, we obtain that the x'_{i1}/x'_{i2} 's are the eigenvalues (over \mathbb{Q}) of $\mathbf{W}_1 \cdot \mathbf{W}_2^{-1}$. These may be computed in polynomial-time from $\mathbf{W}_1 \cdot \mathbf{W}_2^{-1}$ (e.g., by factoring the characteristic polynomial). We hence obtain the x'_{i1}/x'_{i2} 's, for all $i \in [n]$, possibly in a permuted order. We write the fraction x'_{i1}/x'_{i2} as x''_{i1}/x''_{i2} , with co-prime x''_{i1} and x''_{i2} . At this stage, we have the (x''_{i1}, x''_{i2}) 's at hand, for all $i \in [n]$. For each of these pairs, we compute:

$$\gcd(x''_{i1} \cdot x'_2 - x''_{i2} \cdot x'_1, x_0).$$

The prime p_i is a common factor of both $x''_{i1} \cdot x'_2 - x''_{i2} \cdot x'_1$ and x_0 . As all the other factors of x_0 are huge, there is a negligible probability that the gcd is not exactly p_i : another p_j divides $x''_{i1} \cdot x'_2 - x''_{i2} \cdot x'_1$ if and only if $x'_{i1} \cdot x'_{j2} = x'_{i2} \cdot x'_{j1}$.

3.3 Disclosing all the Secret Quantities

At this stage, we know all the p_i 's.

Let $j \in [\tau]$. We have $x_j/y = r_{ij}g_i/(r_{ij}g_i + 1) \pmod{p_i}$. As the numerator and denominator are coprime and very small compared to p_i , they can be recovered by rational reconstruction. We hence obtain $r_{ij}g_i$ for all j . The gcd of the $(r_{ij}g_i)$'s reveals g_i . As a result, we can also recover all the r_{ij} 's and r_i 's.

As $x_1 = r_{i1}g_i/z \pmod{p_i}$ and as the numerator is known, we can recover $z \pmod{p_i}$ for all i , and hence $z \pmod{x_0}$. The h_{ij} 's can then be recovered as well. So can the r'_{ij} 's and a_{ij} 's.

Acknowledgments. The authors thank Michel Abdalla, Jean-Sébastien Coron, Shai Halevi, Adeline Langlois, Tancrede Lepoint, Benoît Libert, Alon Rosen, Gilles Villard and Joe Zimmerman for helpful discussions. The first four author were supported by the

National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2014R1A2A 1A11050917). The last author was supported by the ERC Starting Grant ERC-2013-StG-335086-LATTAC.

References

- [ABP14] Abdalla, M., Benhamouda, F., Pointcheval, D.: Disjunctions for hash proof systems: New constructions and applications. IACR Cryptology ePrint Archive **2014**, 483 (2014)
- [Att14] Attrapadung, N.: Fully secure and succinct attribute based encryption for circuits from multi-linear maps. IACR Cryptology ePrint Archive **2014**, 772 (2014)
- [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)
- [BGN05] Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
- [BLMR13] Boneh, D., Lewi, K., Montgomery, H., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg (2013)
- [BP13] Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: New models and constructions. IACR Cryptology ePrint Archive **2013**, 833 (2013)
- [BR13] Brakerski, Z., Rothblum, G.N.: Obfuscating conjunctions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 416–434. Springer, Heidelberg (2013)
- [BS02] Boneh, D., Silverberg, A.: Applications of multilinear forms to cryptography. Contemporary Mathematics **324**, 71–90 (2002)
- [BWZ14] Boneh, D., Wu, D.J., Zimmerman, J.: Immunizing multilinear maps against zeroizing attacks. IACR Cryptology ePrint Archive **2014**, 930 (2014)
- [CLT13] Coron, J.-S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 476–493. Springer, Heidelberg (2013)
- [CLT14a] Coron, J.-S., Lepoint, T., Tibouchi, M.: Cryptanalysis of two candidate fixes of multilinear maps over the integers. IACR Cryptology ePrint Archive **2014**, 975 (2014)
- [CLT14b] Coron, J.-S., Lepoint, T., Tibouchi, M.: Personal communication (2014)
- [DGHV10] van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
- [Gen09] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of STOC, pp. 169–178. ACM (2009)
- [GGH13a] Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 1–17. Springer, Heidelberg (2013)

- [GGH+13b] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: Proceedings of FOCS, pp. 40–49. IEEE Computer Society Press (2013)
- [GGH14] Gentry, C., Gorbunov, S., Halevi, S.: Graded multilinear maps from lattices. IACR Cryptology ePrint Archive **2014**, 645 (2014)
- [GGHZ14a] Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Fully secure attribute based encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/622 (2014)
- [GGHZ14b] Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666 (2014)
- [GHMS14] Gentry, C., Halevi, S., Maji, H.K., Sahai, A.: Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. IACR Cryptology ePrint Archive **2014**, 929 (2014)
- [GLSW14] Gentry, C., Lewko, A.B., Sahai, A., Waters, B.: Indistinguishability obfuscation from the multilinear subgroup elimination assumption. IACR Cryptology ePrint Archive **2014**, 309 (2014)
- [GLW14] Gentry, C., Lewko, A., Waters, B.: Witness encryption from instance independent assumptions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 426–443. Springer, Heidelberg (2014)
- [LMR14] Lewi, K., Montgomery, H., Raghunathan, A.: Improved constructions of PRFs secure against related-key attacks. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 44–61. Springer, Heidelberg (2014)
- [LS14] Lee, H.T., Seo, J.H.: Security analysis of multilinear maps over the integers. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 224–240. Springer, Heidelberg (2014)
- [Sco02] Scott, M.: Authenticated ID-based key exchange and remote log-in with simple token and PIN number. IACR Cryptology ePrint Archive **2002**, 164 (2002)
- [Sto09] Storjohann, A.: Integer matrix rank certification. In: Proceedings of ISSAC, pp. 333–340. ACM (2009)
- [Zha14] Zhandry, M.: Adaptively secure broadcast encryption with small system parameters. IACR Cryptology ePrint Archive **2014**, 757 (2014)
- [Zim14] Zimmerman, J.: How to obfuscate programs directly. IACR Cryptology ePrint Archive **2014**, 776 (2014)

Honorable Mentions

Robust Authenticated-Encryption AEZ and the Problem That It Solves

Viet Tung Hoang^{1,2(✉)}, Ted Krovetz³, and Phillip Rogaway⁴

¹ Department of Computer Science, University of Maryland, College Park, USA
tvhoang@umd.edu

² Department of Computer Science, Georgetown University, Washington DC, USA

³ Department of Computer Science, California State University, Sacramento, USA

⁴ Department of Computer Science, University of California, Davis, USA

Abstract. With a scheme for *robust* authenticated-encryption a user can select an arbitrary value $\lambda \geq 0$ and then encrypt a plaintext of any length into a ciphertext that's λ characters longer. The scheme must provide all the privacy and authenticity possible for the requested λ . We formalize and investigate this idea, and construct a well-optimized solution, AEZ, from the AES round function. Our scheme encrypts strings at almost the same rate as OCB-AES or CTR-AES (on Haswell, AEZ has a peak speed of about 0.7 cpb). To accomplish this we employ an approach we call *prove-then-prune*: prove security and then instantiate with a *scaled-down* primitive (e.g., reducing rounds for blockcipher calls).

Keywords: AEZ · Authenticated encryption · CAESAR competition · Misuse resistance · Modes of operation · Nonce reuse · Prove-then-prune · Robust AE

1 Introduction

We expose the low cost and high benefit of building authenticated-encryption (AE) schemes that achieve the unprecedentedly strong goal we call *robust* AE (henceforth RAE). We explain why RAE is desirable, define its syntax and security, and explore its guarantees. Then we construct an RAE scheme, AEZ, from AES4 and AES10 (four- and ten-round AES). AEZ's efficiency—nearly that of AES-based OCB [32] or CTR mode—flies in the face of a community's collective work [4, 11–13, 22–25, 35, 38–40, 52–54, 60] in which wide-block enciphering schemes—a special case of RAE—were always far more expensive than conventional blockciphers. Achieving this efficiency has entailed using a design paradigm, the *prove-then-prune* approach, with implications beyond AE.

CIPHERTEXT EXPANSION. One can motivate RAE from a syntactic point of view. Recall that in a nonce-based AE scheme, a plaintext M is mapped to a ciphertext $C = \mathcal{E}_K^{N,A}(M)$ under the control of a key K , nonce N , and associated data (AD) A . Typically the *ciphertext expansion* (or *stretch*) $\lambda = |C| - |M|$ is a constant or user-selectable parameter. For conventional AE, the stretch mustn't

be too small, as customary definitions would break: a trivial adversary can get large advantage. This is because AE definitions “give up” when the first forgery occurs. The issue isn’t *only* definitional: no prior AE scheme provides a desirable security guarantee when the ciphertext expansion is small.

Still, we know that meaningful security is possible even for zero-stretch: a strong pseudorandom permutation buys significant security, even from an AE point of view [5]. What is more, it would seem to be *useful* to allow small stretch, as, for example, short tags can save significant energy in resource-constrained environments (as discussed, e.g., by Struik [58]).

RAE takes a liberal approach towards ciphertext expansion, accommodating whatever stretch a user requests. This leads to schemes that deliver more than conventional AE even when the stretch is not small. Indeed we could have motivated RAE without considering small- λ , describing a desire to achieve nonce-reuse misuse-resistance [51], to automatically exploit novelty or redundancy in plaintexts [5], or to accommodate the release of unverified plaintexts [1, 21]. But our ideas are most easily understood by asking what it means, and what it takes, to do well for any stretch.

DEFINING RAE. So consider an AE scheme that expands a plaintext $M \in \{0, 1\}^*$ by a user-selectable number of bits¹ $\tau \geq 0$. We ask: what’s the best privacy and authenticity guarantee possible for some arbitrary, specified τ ? Robust AE formalizes an answer.

Recall the definition of a *pseudorandom-injection* (PRI) [51]: for each nonce N and associated data A , for a fixed $\tau \geq 0$, the scheme’s encryption algorithm should resemble a uniformly chosen injective function $\pi_{N, A, \tau}$ from binary strings to τ -bit longer ones. Decryption of an invalid ciphertext (one lacking a preimage under π) should return an indication of invalidity.

PRIs were introduced as an alternative characterization of nonce-reuse misuse-resistant AE (henceforth MRAE). But PRIs only approximate MRAE schemes with large stretch. We recast the PRI notion as prescriptive: the user selects $\tau \geq 0$ and then the scheme must look like a PRI for the chosen value. This is our basic definition for RAE.

RAE can be thought of as a bridge connecting blockciphers and AE. When $\tau = 0$ an RAE scheme is a kind of blockcipher—a tweakable blockcipher (TBC) [34] that operates on messages and tweaks of arbitrary length and is secure as strong pseudorandom permutation (PRP). The nonce and AD comprise the tweak. When $\tau \gtrsim 128$ an RAE scheme amounts to an MRAE scheme. An RAE scheme encompasses both objects, and everything in between.

In defining RAE we are actually a bit more generous than what was sketched above, allowing an RAE’s decryption algorithm to return information about an invalid ciphertext beyond a single-valued indication of invalidity. The information just needs to be harmless. To formalize this the reference experiment uses

¹ We’ll later permit arbitrary alphabets. To avoid confusion, we use λ to measure ciphertext expansion in characters (bits, bytes, etc.) and τ to measure it in bits.

a simulator S to provide responses to invalid decryption queries. It must do this without benefit of the family of random injections.

ENCIPHERING-BASED AE. We can achieve RAE with *enciphering-based AE*. The idea, rooted in folklore, was formalized by Bellare and Rogaway [5] and, in a different form, by Shrimpton and Terashima [56]. In its modern incarnation, enciphering-based AE works like this:

Take the message you want to encrypt, augment it with τ bits of redundancy, and then encipher the resulting string by applying an arbitrary-input-length tweakable blockcipher. Tweak this using the nonce, AD, and an encoding of τ . On decryption, check for the presence of the anticipated redundancy and reject a ciphertext if it is not there.

We will prove that this method achieves RAE. In fact, we'll prove that this is so even if the decryption algorithm releases candidate plaintexts with incorrect redundancy.

AEZ. We construct a highly optimized RAE scheme, AEZ. We use the same name to refer to the arbitrary-input-length tweakable blockcipher from which it's built.² With the increasing ubiquity of hardware AES support, we choose to base AEZ on the AES round function.

How AEZ works depends on the length of the input; see Fig. 1. To encipher a plaintext of fewer than 32 bytes we use AEZ-tiny, a balanced-Feistel scheme with a round function based on AES4, a four-round version of AES. The construction builds on FFX [6, 17]. The more interesting case, AEZ-core, is used to encipher strings of 32 bytes or more. It builds on EME [22, 24] and OTR [36]. Look ahead to the top-left panel of Fig. 7. There are two enciphering layers, with consecutive pairs of blocks processed together using a two-round Feistel network. The round function for this is again based on AES4. The mask injected as the middle layer is determined, for each pair of consecutive blocks, using another AES4 call.

PERFORMANCE. AEZ-core is remarkably fast; as the description above implies, we need about five AES4 calls to encipher each consecutive pair of blocks, so ten AES rounds per block. Thus our performance approaches that of CTR-AES. An implementation of AEZ on Haswell using AES-NI has a peak speed of 0.72 cpb—about the same as OCB [32]. Look ahead to Fig. 8. Additionally, invalid strings can be rejected, and AD processed, in about 0.4 AES-equivalents per block, or 0.29 cpb peak (again on Haswell). Only the forward direction of AES is used, saving chip area in hardware realizations. The context size, about 128 bytes, is small, and key setup, about 1.2 AES-equivalents for a 128-bit key, is fast.

For a two-pass mode achieving MRAE, the cluster of performance characteristics described is unexpected. Part of the explanation as to how this is possible lies in the use of a design approach that benefits from both classical and provable-security design. Let us explain.

² Since an RAE scheme trivially determines an arbitrary-input-length tweakable blockcipher (set $\tau = 0$) it makes sense to use a single name for both objects.

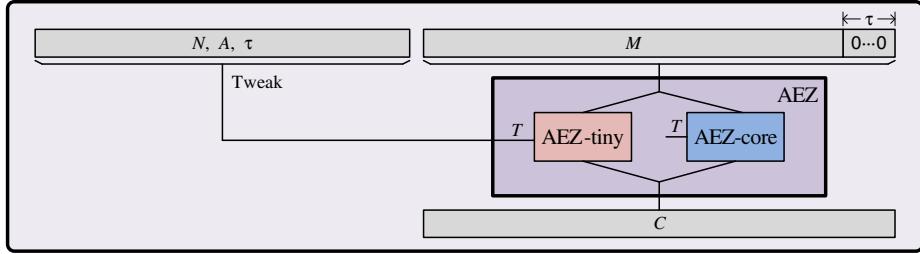


Fig. 1. High-level structure of AEZ. After appending to the message a block of τ zero bits we encipher it using a tweak T comprising the nonce N , associated data A , and stretch τ . Enciphering depends on the length of the plaintext: usually we use AEZ-core, but strings shorter than 32 bytes are enciphered by AEZ-tiny. Both depend on the underlying key K , which is not shown in the diagram above.

PROVE-THEN-PRUNE DESIGN. We designed AEZ using an approach we call *prove-then-prune*. It works like this:

To achieve some complex cryptographic goal, design a scheme in the provable-security tradition, choosing an underlying primitive and demonstrably achieving the goal when it's instantiated by an object achieving some standard assumption. Then, to improve speed, selectively instantiate some of the applications of the primitive using a *scaled-down* (e.g., reduced-round) construction. Use heuristic or cryptanalytic reasons to support the expectation that, despite scaling down, the scheme remains secure.

Specifically, AEZ is designed in terms of a tweakable blockcipher (TBC). If this TBC had been instantiated in the “usual” way, say using AES and the XE construction [34, 49], we would have a provably-sound design on message space $\{0, 1\}^{\geq 128}$. The cost would be about 2.5 times the cost of AES. But to speed things up, we instantiate most TBC calls with an AES4-based construction. Heuristics reasons to suggest that security nonetheless remains. Our design was specifically chosen so as to make a scaled-down instantiation plausible.

The thesis underlying prove-then-prune approach is that it can be instrumental for devising highly efficient schemes for complex aims. We believe that if the instantiation is done judiciously, then the scaled-down scheme retains *some* assurance benefit. Still, it is important to emphasize the limitations of prove-then-prune. Naming an approach is not license to abuse it. The method is dangerous in the same sort of way that designing a confusion/diffusion primitive is: one has no guarantees for the object that will actually be used. Additionally, the set of people with provable-security competence is nearly disjoint from those with cryptanalytic competence. The authors think it essential that cryptanalysts study AEZ. This is all the more true because pruning was aggressive.

In some way, prove-then-prune is implicit in prior work: schemes like ALRED [15] typify a trend in which reduced-round AES is used in contexts where full AES would demonstrably do the job.

RAE BENEFITS. What do we hope to gain with RAE? Our definition and scheme are meant to achieve all of the following: (1) If (M, A) tuples are known *a priori* not to repeat, no nonce is needed to ensure semantic security. (2) If there's redundancy in plaintexts whose presence is verified on decryption, this augments authenticity. (3) Any authenticator-length can be selected, achieving best-possible authenticity for this amount of stretch. (4) Because of the last two properties, one can minimize length-expansion in many bandwidth-constrained applications. (5) If what's supposed to be a nonce should accidentally get repeated, the privacy loss is limited to revealing repetitions in (N, A, M) tuples, while authenticity is not damaged at all. (6) If a decrypting party leaks some or all of a putative plaintext that was supposed to be squelched because of an authenticity-check failure, this won't compromise privacy or authenticity.

The authors believe that the properties enumerated would sometimes be worth a considerable computational price. Yet the overhead we pay is low: AEZ is almost as fast as OCB.

DISCUSSION. AEZ's name is meant to simultaneously suggest AE, AES, and EZ (easy), the last in the sense of ease of correct use. But the simplicity is for the user; we would not claim that the AEZ algorithm is simple.

Since McOE and COPA [2, 20], some recent AE schemes have been advertised as nonce-reuse misuse-resistant despite being online.³ But online schemes are never misuse-resistant in the sense originally defined [51].⁴ They never support automatic exploitation of novelty or verified redundancy [5] and are always vulnerable to a simple message-recovery attack [47]. We disagree with the presumption that two-pass AE schemes are routinely problematic; in fact, our work suggests that, on capable platforms, there isn't even a performance penalty. Finally, short messages routinely dominate networking applications, and we know of no application setting where it's important to limit latency to just a few bytes, the implicit expectation for proposed online schemes.

This paper upends some well-entrenched assumptions. Before, AE-quality was always measured with respect to an aspirational goal; now we're suggesting to employ an achievable one. Before, substantial ciphertext expansion was seen as necessary for any good AE; now we're allowing an arbitrary, user-supplied input. Before, AE schemes and blockciphers were considered fundamentally different species of primitives; now we're saying that, once the definitions are strengthened, they're pretty much the same thing. Before, one could either give a provable-security design or one that follows a more heuristic tradition; now we're doing the one and yet still finding need for the other.

AEZ is one of 57 CAESAR submissions [7]. It's distinguished by being the notionally strongest submission. We expect it to help clarify the potential cost and benefit of two-pass AE.

³ By *online* we mean that the encryption algorithm can be realized in $O(1)$ memory and a single pass over M .

⁴ If the first bit of ciphertext doesn't depend on the last bit of plaintext an adversary easily wins the MRAE game.

2 Prior AE Definitions

Fix an alphabet Σ . Typically Σ is $\{0, 1\}$ or $\{0, 1\}^8$, but other values, like $\Sigma = \{0, 1, \dots, 9\}$, are fine. For $x \in \Sigma^*$ let $|x|$ denote its length. We write ε for the empty string and $x \leftarrow X$ for uniformly sampling from a distribution X . If X is a finite set, it has the uniform distribution.

SYNTAX. We formalize a nonce-based AE scheme as a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key space \mathcal{K} is a set of strings with an associated distribution. The encryption algorithm \mathcal{E} is deterministic and maps a four-tuple $(K, N, A, M) \in (\Sigma^*)^4$ to a value $C = \mathcal{E}_K^{N,A}(M)$ that is either a string in Σ^* or the distinguished symbol \perp . Later we will allow AD to be a vector of strings, $A \in (\Sigma^*)^*$. The distinction is insignificant insofar as we can always encode a vector of strings as a string. We require the existence of sets \mathcal{N} , \mathcal{A} and \mathcal{M} (the nonce space, AD space, and message space) such that $\mathcal{E}_K^{N,A}(M) \neq \perp$ iff $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$. The decryption algorithm \mathcal{D} is deterministic and takes a four-tuple (K, N, A, C) to a value $\mathcal{D}_K^{N,A}(C) \in \Sigma^* \cup \{\perp\}$. The length of a string-valued $C = \mathcal{E}_K^{N,A}(M)$ is not allowed to depend on anything beyond $|N|$, $|A|$ and $|M|$. In fact, usually $\lambda = |C| - |M|$ is a constant, in which case we call the scheme λ -expanding and refer to λ as the *ciphertext expansion* or *stretch*. We require that if $C = \mathcal{E}_K^{N,A}(M)$ is a string then $\mathcal{D}_K^{N,A}(C) = M$. Algorithm \mathcal{D} rejects ciphertext C if $\mathcal{D}_K^{N,A}(C) = \perp$ and accepts it otherwise.

AE AND MRAE SECURITY. Both conventional-AE and MRAE security can be defined using a compact, all-in-one formulation [51]. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE-scheme. Consider an adversary \mathcal{A} with access to an encryption oracle Enc and a decryption oracle Dec . We define the MRAE security of \mathcal{A} as $\mathbf{Adv}_{\Pi}^{\text{mrae}}(\mathcal{A}) = \Pr[\mathcal{A}^{\text{Real}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\text{Ideal}_{\Pi}} \Rightarrow 1]$, the difference in the probability that \mathcal{A} outputs 1 when run in the **Real** and **Ideal** games of Fig. 2. Both begin by selecting $K \leftarrow \mathcal{K}$. Game **Real** answers encryption queries (N, A, M) with $\mathcal{E}_K^{N,A}(M)$ and decryption queries (N, A, C) with $\mathcal{D}_K^{N,A}(C)$. Game **Ideal** answers $\text{Dec}(N, A, C)$ queries with \perp and $\text{Enc}(N, A, M)$ queries with $|C|$ uniformly chosen characters, where $C \leftarrow \mathcal{E}_K^{N,A}(M)$. For games **Real** and **Ideal**, adversaries may not repeat an Enc or Dec query, ask an Enc query $(N, A, M) \notin \mathcal{N} \times \mathcal{A} \times \mathcal{M}$, ask a Dec query $(N, A, C) \notin \mathcal{N} \times \mathcal{A} \times \Sigma^*$, or ask a Dec query (N, A, C) after an Enc query of (N, A, M) returned C .

The above definition captures MRAE security because repeated nonces were allowed and were properly serviced. For the conventional AE notion, $\mathbf{Adv}_{\Pi}^{\text{ae}}(\mathcal{A})$, modify **Real** and **Ideal** by having an $\text{Enc}(N, A, M)$ query following an earlier $\text{Enc}(N, A', M')$ query return \perp . This has the same effect as prohibiting repeated N -values to the Enc oracle.

PRI SECURITY. We define security in the sense of a *pseudorandom-injection* (PRI) [51]. Fix a λ -expanding AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$; for now, λ is a constant associated to a (well-behaved) AE scheme. Let $\mathbf{Adv}_{\Pi}^{\text{pri}}(\mathcal{A}) = \Pr[\mathcal{A}^{\text{Real}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\text{PRI}_{\Pi}} \Rightarrow 1]$ with the oracles again defined in Fig. 2. There $\text{Inj}(\lambda)$ denotes

initialize	Ideal_{II}	initialize	Real_{II}	initialize	PRI_{II}
$K \leftarrow \mathcal{K}$		$K \leftarrow \mathcal{K}$		$\text{for } (N, A) \in \mathcal{N} \times \mathcal{A} \text{ do}$	
oracle Enc (N, A, M)		oracle Enc (N, A, M)		$\pi_{N,A} \leftarrow \text{Inj}(\lambda)$	
$C' \leftarrow \mathcal{E}_K(N, A, M)$		return $\mathcal{E}_K(N, A, M)$		oracle Enc (N, A, M)	
$C \leftarrow \Sigma^{ C' }$				return $\pi_{N,A}(M)$	
return C				oracle Dec (N, A, C)	
oracle Dec (N, A, C)		oracle Dec (N, A, C)		if $\exists M \in \mathcal{M}$ s.t. $\pi_{N,A}(M) = C$	
return \perp		return $\mathcal{D}_K(N, A, C)$		then return M	
				return \perp	
initialize	REAL_{II}	initialize		RAE_{II} and RAE_{II,S}	
$K \leftarrow \mathcal{K}$				for $(N, A, \lambda) \in \Sigma^* \times \Sigma^* \times \mathbb{N}$ do	
oracle Enc (N, A, λ, M)				$\pi_{N,A,\lambda} \leftarrow \text{Inj}(\lambda)$	
return $\mathcal{E}_K(N, A, \lambda, M)$				$\theta \leftarrow \varepsilon$	
oracle Dec (N, A, λ, C)		oracle Enc (N, A, λ, M)			
return $\mathcal{D}_K(N, A, \lambda, C)$		return $\pi_{N,A,\lambda}(M)$			
		oracle Dec (N, A, λ, C)			
		if $\exists M \in \mathcal{M}$ s.t. $\pi_{N,A,\lambda}(M) = C$ then return M			
		$M \leftarrow \perp$			————— for RAE_{II}
		$(M, \theta) \leftarrow S(N, A, \lambda, C, \theta)$			————— for RAE_{II,S}
		return M			

Fig. 2. Games for defining security. The top three games are the usual ones for defining the AE and MRAE notions. The bottom two games are used to define RAE.

the set of all one-to-one functions from Σ^* to Σ^* that increase the length of their inputs by λ characters. The same query restrictions apply as before.

Besides defining PRI security, Rogaway and Shrimpton showed that, for large ciphertext expansion λ , the notion essentially coincides with MRAE security [51]. Below we clarify the role of the ciphertext expansion by giving a sharper extended version of their result. To state our bound, define the *misuse count* as follows. Initially, set $r = 0$. Then, for each encryption query $\text{Enc}(N, A, M)$, if there was a prior query (N, A, M') such that $|M'| = |M|$, increment r by 1. The final value of r is the misuse count. Below we show that good PRI security implies good MRAE security as long as q is small compared to $|\Sigma|^\lambda$ and r is small compared to $|\Sigma|^{(\lambda+m_{\min})/2}$ (with all variables defined below). The proof is in Appendix B.1.

Theorem 1. $|\text{Adv}_{\Pi}^{\text{pri}}(\mathcal{A}) - \text{Adv}_{\Pi}^{\text{mrae}}(\mathcal{A})| \leq 2q/|\Sigma|^\lambda + (r^2 + r)/|\Sigma|^{\lambda+m_{\min}+1}$ for any λ -expanding AE scheme Π and adversary \mathcal{A} , where r is the misuse count of \mathcal{A} 's queries, q is the number of queries it asks, and m_{\min} is the length of the shortest string in the message space.

In short, the PRI definition captures best-possible security of a λ -expanding AE scheme, while the MRAE formulation captures an unreachable ideal. The gap between the realizable and the ideal is formalized by Theorem 1. It is small if the ciphertext expansion is large, and it is large if the ciphertext expansion

is small. The latter is so because any *actual* encryption algorithm must map distinct (N, A, M) and (N, A, M') to distinct ciphertexts, whence real encryption *can't* return uniformly random characters. Similarly, for any infinite message space, *some* unqueried ciphertexts *must* be valid, whence a decryption oracle that *always* returns an indication of invalidity is hoping for too much. Building on the PRI notion, we will now look towards an even more precise way to capture best-possible AE security.

3 RAE Security

SYNTAX. The principle difference between a PRI and an RAE scheme is that, for the latter, the ciphertext expansion λ is no longer a property of a scheme: it's an arbitrary *input* from the user. All values $\lambda \in \mathbb{N}$ should be allowed.⁵ Corresponding to this change, we'll write $\mathcal{E}_K^{N,A,\lambda}(M)$ and $\mathcal{D}_K^{N,A,\lambda}(C)$. The difference may look small, but its consequences are not.

Fix an alphabet Σ . Our formal definition again has an RAE scheme being a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, but with the signature of \mathcal{E} and \mathcal{D} updated. The encryption algorithm \mathcal{E} is deterministic and maps a five-tuple $(K, N, A, \lambda, M) \in (\Sigma^*)^3 \times \mathbb{N} \times \Sigma^*$ to a string $C = \mathcal{E}_K^{N,A,\lambda}(M)$ of length $|M| + \lambda$. For maximal utility when realized, we are not permitting a return value of \perp : an RAE scheme must be able to encrypt any M using any N , A , and λ . The decryption algorithm \mathcal{D} is deterministic and takes a five-tuple (K, N, A, λ, C) to a value $\mathcal{D}_K^{N,A,\lambda}(C) \in \Sigma^* \cup \{\perp\}$. We require that $\mathcal{D}_K^{N,A,\lambda}(\mathcal{E}_K^{N,A,\lambda}(M)) = M$ for all K, N, A, λ, M . If there's no M such that $C = \mathcal{E}_K^{N,A,\lambda}(M)$ then $\mathcal{D}_K^{N,A,\lambda}(C) = \perp$. Later in this section we will relax this requirement as a way to model the possibility of decryption algorithms that reveal information beyond an indication of invalidity.

RAE SECURITY. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an RAE scheme over alphabet Σ . Its security is defined using the games **REAL** $_{\Pi}$ and **RAE** $_{\Pi}$ at the bottom of Fig. 2. (For the moment, ignore **RAE** $_{\Pi,S}$.) The adversary \mathcal{A} has two oracles, an encryption oracle Enc and a decryption oracle Dec. For game **REAL**, these are realized by the actual encryption and decryption algorithms, which now take in the argument λ . For game **RAE** $_{\Pi}$ we behave according to the family of random injections $\pi_{N,A,\lambda}$ chosen at the beginning of the game, responding to each encryption query (N, A, λ, M) with $C = \pi_{N,A,\lambda}(M)$ and responding to each decryption query (N, A, λ, C) with $\pi_{N,A,\lambda}^{-1}(C)$, if that inverse exists, and \perp if it does not. We let $\mathbf{Adv}_{\Pi}^{\text{rae}}(\mathcal{A}) = \Pr[\mathcal{A}^{\text{REAL}}_{\Pi} \Rightarrow 1] - \Pr[\mathcal{A}^{\text{RAE}}_{\Pi} \Rightarrow 1]$. There are no restrictions on the kinds of queries the adversary may make.

To gain some appreciation for the RAE definition, consider an adversary that asks to encrypt a message M using a single byte of stretch. Such a scheme would not be considered secure in the MRAE setting, as forging with probability $1/256$ is easy. But under the RAE viewpoint, that isn't a defect per se, as the user who

⁵ It might be OK to set some reasonable upperbound $\lambda \leq \lambda_{\max}$, but there shouldn't be a nonzero lowerbound.

requests one-byte expansion would *expect* 1/256 of all ciphertexts to have some preimage. If a user should try to decrypt such a ciphertext C using the same K, N, A but $\lambda = 0$, a plaintext *will* emerge, never an indication of invalidity, but that plaintext should be unrelated to the originally encrypted one.

DECRYPTION-CALL LEAKAGE. An AE scheme will fail to approximate the **RAE $_{\Pi}$** abstraction if its decryption algorithm, when presented an invalid ciphertext, routinely returns anything beyond an indication of invalidity. We now explain how to relax this expectation so that it's OK to return additional material as long as it is known to be useless.

We said earlier that, for an RAE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and any N, A, λ, C , if there's no M such that $C = \mathcal{E}_K^{N, A, \lambda}(M)$ then we expect $\mathcal{D}_K^{N, A, \lambda}(C)$ to return \perp . Let us relax this requirement so that $\mathcal{D}_K^{N, A, \lambda}(C)$ may instead return a string, as long as its length is not $|C| - \lambda$. Any such string is trivially recognized as invalid, so, in effect, we are having \mathcal{D} return both \perp and an arbitrary piece of side information Y . We are not suggesting that the “real” decryption algorithm should return anything other than \perp when presented an invalid ciphertext; instead, we are effectively overloading \mathcal{D} by folding into it a “leakage function” that captures that which a decryption algorithm's realization may leak about a presented ciphertext.

Using this generalized syntax, we define a game **RAE $_{\Pi, S}$** parameterized by a probabilistic algorithm S , the *simulator*. Again see Fig. 2. Simulator S is called upon to produce imitation ciphertexts when there's no preimage under $\pi_{N, A, \lambda}$. To accomplish this task S is provided nothing beyond the current oracle query and any saved state θ it wants to maintain. An RAE scheme is judged secure if there's a simulator S —preferably an efficient one—such that $(\mathcal{E}, \mathcal{D})$ is indistinguishable from the pair of oracles defined in **RAE $_{\Pi, S}$** . We refine the RAE advantage by asserting that $\mathbf{Adv}_{\Pi, S}^{\text{rae}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathbf{REAL}_{\Pi}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{RAE}_{\Pi, S}} \Rightarrow 1]$. The “basic” RAE definition corresponds to the case where simulator S ignores its input and returns (\perp, ε) .

The RAE definition effectively captures that, while it may be “nice” for decryption to reveal nothing but \perp on presentation of an invalid ciphertext, there are plenty of other things we could return without damaging privacy or authenticity. In fact, it really doesn't matter *what* is returned just so long as it's recognizably invalid and doesn't depend on the key.

ILLUSTRATION. Fig. 3 illustrates two possibilities for how an RAE scheme might encrypt 2-bit strings with 2-bit ciphertext expansion ($\lambda = 2$). The key K , nonce N , and AD A are all fixed. For encryption, the four possible plaintexts are bijectively paired with four of the 16 possible ciphertexts. For decryption we show two possibilities. On the left is a *conventional* decryption algorithm: the 12 ciphertexts without a preimage decrypt to an indication of invalidity. One expects the simulator to always return (\perp, ε) . On the right is a *sloppy* decryption algorithm. The 12 ciphertexts with no preimage decrypt to 12 distinct strings, all recognizably invalid, all of the form $abcd \in \{0, 1\}^4$ with $cd \neq 00$. Here the simulator S might sample without replacement from the named set of size 12.

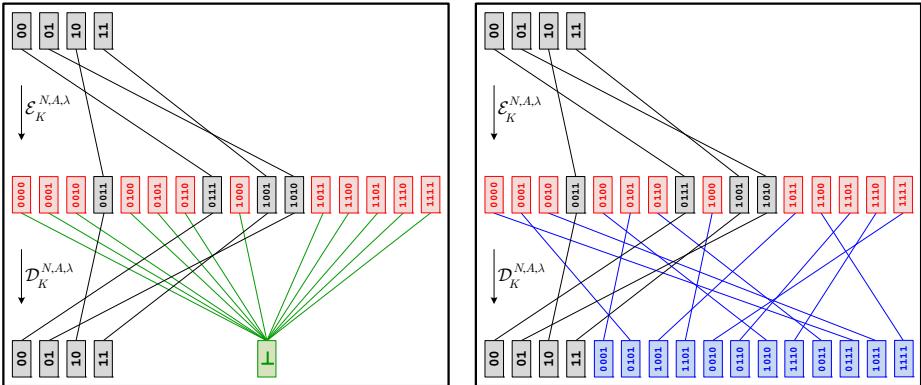


Fig. 3. Illustrating RAE. Two ways an RAE scheme might encrypt and decrypt a 2-bit string with 2-bit stretch.

DISCUSSION. The reader may have noticed that there is no distinction in the RAE security definition between the nonce N and associated data (AD) A . For this reason, either could be dropped—say the nonce—leaving us a signature $\mathcal{E}_K^{A,\lambda}(M)$ and $\mathcal{D}_K^{A,\lambda}(C)$. There’s an especially good argument for doing this when the AD A is vector-valued: the user is already free to use one of its components as a nonce. Still, for greater uniformity in treatment across AE notions, and to encourage users to provide a nonce, we have retained both N and A .

We gave our definition of RAE into two stages only for pedagogical purposes: this paper offers only one definition for RAE. The simulator S may be trivial or not; that is the only distinction.

Andreeva *et. al* [1] recently provided several security definitions also meant to capture the requirement that a decryption algorithm releases only harmless information when presented an invalid ciphertext and a repeated nonce. Our own work is radically different from theirs insofar as we provide a single definition, RAE, that rolls into it this, among many, considerations.

4 Verified Redundancy Enhances Authenticity

If a plaintext contains redundancy, one naively expects that verifying its presence upon decryption should enhance the authenticity guarantee provided. For the case of enciphering-based encryption, which provides no authenticity guarantee on its own, this has been formally supported [5, 51]. But even in this case the existing results are with respect to conventional notions of AE, and such notions are too blunt to capture what one expects from verified redundancy. This is because the notions “give up” as soon as a single ciphertext forgery is made.

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be RAE scheme and let $v: \Sigma^* \rightarrow \{0, 1\}$ be a function for indicating the “valid” strings: it determines $\mathcal{M}_v \subseteq \Sigma^*$ by $\mathcal{M}_v = \{M \in \Sigma^*: v(M) = 1\}$. Let $\Pi_v = (\mathcal{K}, \mathcal{E}, \tilde{\mathcal{D}})$ be the AE scheme built from Π that declares messages invalid if v says so: $\tilde{\mathcal{D}}_K^{N,A,\lambda}(C) = M$ if $|M| = |C| - \lambda$ and $v(M) = 1$, or

if $|M| \neq |C| - \lambda$, where $M = \mathcal{D}_K^{N,A,\lambda}(C)$, while $\tilde{\mathcal{D}}_K^{N,A,\lambda}(C) = \mathbf{0} \parallel M$ otherwise, with $\mathbf{0}$ a canonical point in Σ . Let $d_v = \max_{\ell \in \mathbb{N}} \{(|\mathcal{M}_v \cap \Sigma^\ell|)/|\Sigma|^\ell\}$ be the density of \mathcal{M}_v .

Suppose, for example, that $\Sigma = \{0, 1\}$ and $d_v = 1/256$: there's a byte worth of redundancy in the message space. We'd like to be able to make statements about the authenticity of Π_v such as: the chance that an adversary can forge 10 successive, distinct ciphertexts is negligibly more than 2^{-80} . Conventional AE definitions don't let one say such a thing; they stop at the bound $q/|\Sigma|^\lambda$ where q is the number of queries and λ is the ciphertext expansion (assumed here to be a constant). One would like to obtain a much sharper bound via d_v and λ —in our example, the forgery probability should be about $q(d_v/|\Sigma|^\lambda)^{10}$. This way, even if, say, $\lambda = 0$ and $d_v = 1/2$, we are *still* able to make strong statements about the security of Π_v . Intuitively, for an RAE scheme Π , the scheme Π_v should have about $(\lambda_{\min} + \log(1/d_v)) \log(|\Sigma|)$ bits of authenticity, where λ_{\min} is the minimum ciphertext expansion of any query—even after multiple successful forgeries and even in the presence of decryption leakage, future forgeries still remain just as hard.

To capture the intuition above, in Theorem 2 we show that Π_v itself is RAE-secure. The proof is in Appendix B.2. Consequently, in game **RAE**, for any query (N, A, λ, C) with $|C| = \ell + \lambda$ to **Dec**, the chance that this query is a successful forgery is about $|\mathcal{M}_v \cap \Sigma^\ell|/|\Sigma|^{\ell+\lambda} \leq d_v/|\Sigma|^\lambda$, despite any decryption leakage and past successful forgeries.

Theorem 2. Let Π and Π_v be defined as above. There is an explicitly given reduction \mathcal{R} with the following property. For any simulator S and any adversary \mathcal{A} , there is a simulator S' such that the adversary $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}_{\Pi,S}^{\text{rae}}(\mathcal{B}) = \mathbf{Adv}_{\Pi_v,S'}^{\text{rae}}(\mathcal{A})$. Adversary \mathcal{B} makes the same queries as \mathcal{A} and has essentially the same running time.

Note that for good RAE security, we want the simulator S to be efficient. This is important for privacy, but when the concern is authenticity, it's less of an issue: a computationally-unbounded simulator may give the adversary some information that it can't compute itself, but as long as the adversary can't forge, whatever the adversary learns from the simulator is irrelevant for authenticity. Still, in the proof of Theorem 2, for each query (N, A, λ, C) , the simulator S' either runs S or samples from $\Sigma^\ell \cap \mathcal{M}_v$, where $\ell = |C| - \lambda$. For functions v that arise from real-world usage, sampling from $\Sigma^\ell \cap \mathcal{M}_v$ is likely to be simple and efficient, whence S' will be about as fast as S itself.

5 Robust AE from an Arbitrary-Input-Length TBC

We now show how to make an AE scheme that achieves RAE security. We begin with some basic definitions. Let $\mathcal{M} \subseteq \Sigma^*$ and \mathcal{T} be sets. A *blockcipher* $\widetilde{\mathbb{E}} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ is a mapping such that $\widetilde{\mathbb{E}}_K^T(\cdot) = \widetilde{\mathbb{E}}(K, T, \cdot)$ is a length-preserving permutation on \mathcal{M} for any K, T . Thus $|\widetilde{\mathbb{E}}_K^T(X)| = |X|$ and there's a unique $\widetilde{\mathbb{D}} : \mathcal{K} \times \mathcal{T} \times \{0, 1\}^* \rightarrow \mathcal{M} \cup \{\perp\}$ such that $\widetilde{\mathbb{E}}_K^T(M) = C$ implies

$\tilde{\mathbb{D}}_K^T(C) = M$ and $\tilde{\mathbb{D}}_K^T(C) = \perp$ when there's no M such that $\tilde{\mathbb{E}}_K^T(M) = C$. We call \mathcal{T} the *tweak space* of $\tilde{\mathbb{E}}$. When $|\mathcal{T}| = 1$ we make the tweak implicit, writing $\mathbb{E}: \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{M}$, now with inverse \mathbb{D} . We define $\text{Perm}(\mathcal{M})$ as the set of all length-preserving permutations on \mathcal{M} , and $\text{Perm}(\mathcal{T}, \mathcal{M})$ the set of all mappings $\tilde{\pi}: \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ where $\tilde{\pi}(T, \cdot) \in \text{Perm}(\mathcal{M})$ for all $T \in \mathcal{T}$. We usually use *encipher* instead of *encrypt* when speaking of applying a blockcipher, and similarly for *decipher* and *decrypt*.

An *arbitrary-input-length* blockcipher is a blockcipher with message space $\mathcal{M} = \Sigma^*$. To be maximally useful, we will want a rich tweak space as well. These are versatile objects. A bit less general, a *wide-block* blockcipher has message space $\Sigma^{\geq n}$ for some fixed n . Again one prefers a rich tweak space. A *conventional* blockcipher has message space $\{0, 1\}^n$ for some fixed n .

The strong, tweakable, PRP advantage of an adversary A attacking a blockcipher $\tilde{\mathbb{E}}$ is defined as $\mathbf{Adv}_{\tilde{\mathbb{E}}}^{\pm\text{PRP}}(\mathcal{A}) = \Pr[K \leftarrow \mathcal{K} : \mathcal{A}^{\tilde{\mathbb{E}}_K(\cdot, \cdot), \tilde{\mathbb{D}}_K(\cdot, \cdot)} \Rightarrow 1] - \Pr[\tilde{\pi} \leftarrow \text{Perm}(\mathcal{T}, \mathcal{M}) : \mathcal{A}^{\tilde{\pi}(\cdot, \cdot), \tilde{\pi}^{-1}(\cdot, \cdot)} \Rightarrow 1]$. We'll write $\mathbf{Adv}_{\mathbb{E}}^{\pm\text{PRP}}(\mathcal{A}) = \Pr[K \leftarrow \mathcal{K} : \mathcal{A}^{\mathbb{E}_K(\cdot), \mathbb{D}_K(\cdot)} \Rightarrow 1] - \Pr[\pi \leftarrow \text{Perm}(\mathcal{M}) : \mathcal{A}^{\pi(\cdot), \pi^{-1}(\cdot)} \Rightarrow 1]$ if there's no tweak. If we prohibit the adversary \mathcal{A} from querying the second oracle we drop the word “strong” and write $\mathbf{Adv}_{\mathbb{E}}^{\text{PRP}}(\mathcal{A})$ and $\mathbf{Adv}_{\mathbb{E}}^{\text{PRP}}(\mathcal{A})$ respectively.

ENCODE-THEN-ENCIPHER. Fix Σ . Let $\tilde{\mathbb{E}}: \mathcal{K} \times \mathcal{T} \times \Sigma^* \rightarrow \Sigma^*$ be an arbitrary input-length tweakable blockcipher with tweak space $\mathcal{T} = \Sigma^* \times \Sigma^* \times \mathbb{N}$. Let \mathbb{D} be its inverse. Let $\text{Encode}: \Sigma^* \times \mathbb{N} \rightarrow \Sigma^*$ be an injective function satisfying $|\text{Encode}(M, \lambda)| = |M| + \lambda$. We write the second argument to Encode as a subscript, $\text{Encode}_\lambda(M) \in \Sigma^{|M|+\lambda}$. An example encoding function is $\text{Encode}_\lambda(M) = M \parallel \mathbf{0}^\lambda$.

For any encoding function Encode there's a corresponding $\text{Decode}: \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \cup \{\perp\}$ such that $\text{Decode}_\lambda(X) = M$ if there's an M satisfying $\text{Encode}_\lambda(M) = X$, while $\text{Decode}_\lambda(X) = \perp$ if there's no such M . We expect Encode and Decode to be trivially computable, as in the example.

From $\tilde{\mathbb{E}}: \mathcal{K} \times \mathcal{T} \times \Sigma^* \rightarrow \Sigma^*$ and Encode we define the encode-then-encipher construction as the RAE scheme $\text{EtE}[\text{Encode}, \tilde{\mathbb{E}}] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where

- $\mathcal{E}_K^{N, A, \lambda}(M) = \tilde{\mathbb{E}}_K^{(N, A, \lambda)}(\text{Encode}_\lambda(M))$,
- $\mathcal{D}_K^{N, A, \lambda}(C) = M$ if $\tilde{\mathbb{D}}_K^{(N, A, \lambda)}(C) = X$ and M satisfies $\text{Encode}_\lambda(M) = X$,
- $\mathcal{D}_K^{N, A, \lambda}(C) = X$ if $\tilde{\mathbb{D}}_K^{(N, A, \lambda)}(C) = X$ and no M satisfies $\text{Encode}_\lambda(M) = X$.

We stress that decryption does not simply return \perp when called on an invalid (N, A, λ, C) , as is conventionally done; instead, we define decryption to “leak” the entire improperly encoded string X . Nonetheless, Theorem 3 shows that $\text{EtE}[\text{Encode}, \tilde{\mathbb{E}}]$ is RAE-secure when $\tilde{\mathbb{E}}$ is secure as a strong, tweakable PRP. Its proof appears in the full version [28].

Theorem 3 (EtE is RAE-secure). Let Encode and $\tilde{\mathbb{E}}: \mathcal{K} \times \mathcal{T} \times \Sigma^* \rightarrow \Sigma^*$ be defined as above. Then there's an explicitly given reduction \mathcal{R} and an efficient simulator S with the following property. For any adversary \mathcal{A} , the adversary $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}_{\text{EtE}[\text{Encode}, \tilde{\mathbb{E}}, S]}^{\text{rae}}(\mathcal{A}) \leq \mathbf{Adv}_{\tilde{\mathbb{E}}}^{\pm\text{PRP}}(\mathcal{B})$. It makes at most q queries whose total length is at most that of \mathcal{A} 's queries plus $q\lambda_{\max}$, where q is the number of \mathcal{A} 's queries and λ_{\max} is the largest stretch among them. The running time of \mathcal{B} is about that of \mathcal{A} , plus the time associated to computations of Encode and Decode .

6 Wide-Block Enciphering: AEZ-core

Let $n \geq 1$ be an integer and let $\{0, 1\}^{\geq 2n} = \{x \in \{0, 1\}^*: |x| \geq 2n\}$. Define the *block length* of a string x as $\lceil |x|/n \rceil$. We show how to build a strong PRP on $\{0, 1\}^{\geq 2n}$ from a TBC on $\{0, 1\}^n$. We'll use about 2.5 TBC calls per n -bit block. Later we'll instantiate the TBC using mostly AES4, employing the prove-then-prune paradigm to selectively scale-down. This will reduce the amortized cost to about one AES call per block. Also see the full version [28] for how to tweak a wide-block blockcipher.

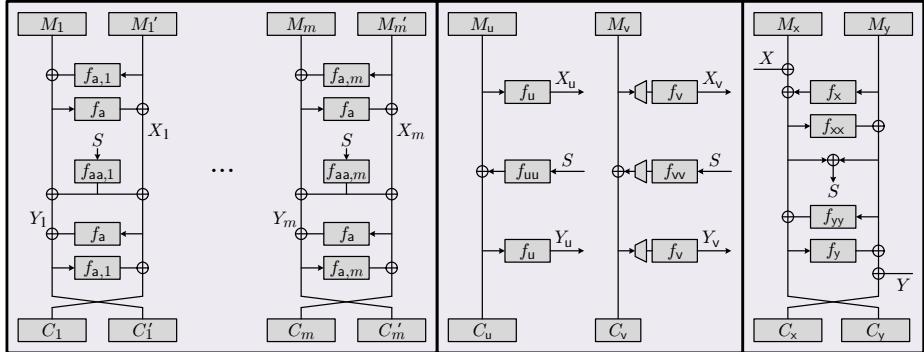
We begin by recalling the definition of a pseudorandom function (PRF) $f: \mathcal{K} \times \mathcal{M} \rightarrow \{0, 1\}^n$. For an adversary \mathcal{A} attacking f , its PRF advantage is $\mathbf{Adv}_f^{\text{prf}}(\mathcal{A}) = \Pr[K \leftarrow \mathcal{K}: \mathcal{A}^{f_K(\cdot)} \Rightarrow 1] - \Pr[\rho \leftarrow \text{Func}(\mathcal{M}, n): \mathcal{A}^{\rho(\cdot)} \Rightarrow 1]$ where $\text{Func}(\mathcal{M}, n)$ is the set of all functions from \mathcal{M} to $\{0, 1\}^n$.

AEZ-CORE. Let $\mathcal{T} = \{\mathbf{a}, \mathbf{u}, \mathbf{uu}, \mathbf{v}, \mathbf{vv}, \mathbf{x}, \mathbf{xx}, \mathbf{y}, \mathbf{yy}\} \cup (\{\mathbf{a}, \mathbf{aa}\} \times \mathbb{N})$ be the tweak space. Suppose we have a PRF $f: \mathcal{K} \times (\mathcal{T} \times \{0, 1\}^n) \rightarrow \{0, 1\}^n$. One can instantiate this with a TBC \tilde{E} on $\{0, 1\}^n$ by setting $f_K(K, (T, X)) = \tilde{E}_K^T(X)$. Consider the wide-block blockcipher AEZ-core[f] defined and illustrated in Fig. 6. It loosely follows EME/EME2 [22, 24, 29], but avoids all doubling operations and only uses the forward direction of the underlying TBC. AEZ-core[f] operates on $\mathcal{M} = \{0, 1\}^{\geq 2n}$ and itself takes in no tweak. Theorem 4 shows that it's a strong PRP. The proof is in the full version [28].

Theorem 4. Let $n \geq 1$ be an integer and let \mathcal{T} and f be as above. There's an explicitly given reduction \mathcal{R} with the following property. For any adversary \mathcal{A} , adversary $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}_{\text{AEZ-core}[f]}^{\pm\text{PRP}}(\mathcal{A}) \leq \mathbf{Adv}_f^{\text{prf}}(\mathcal{B}) + 2\sigma^2/2^n$ where σ is the total block length of \mathcal{A} 's queries. Adversary \mathcal{B} uses the same running time as \mathcal{A} , and makes at most 2.5σ queries.

DISCUSSION. AEZ-core and its inverse are almost the same: the only change needed is to take the rightmost column of tweaks in reverse order. Given that one must have *some* asymmetry in an RAE scheme—an involution is certainly RAE-insecure—this is about as symmetric a design as one could hope for. A high degree of symmetry can help maximize efficiency of both hardware and software. Symmetry is the reason for the wire-crossing just before each $C_i C'_i$.

Among the efficiency characteristics of AEZ-core is that one can selectively decrypt a chosen block about 2.5 times more quickly than decrypting everything.



```

10 algorithm AEZ-core( $K, M$ ) //AEZ-core
11  $M_1M'_1 \dots M_mM'_m M_{uv} M_xM_y \leftarrow M$ 
12 where  $|M_1| = \dots = |M'_m| = |M_x| = |M_y| = n$  and  $|M_{uv}| < 2n$ 
13  $d \leftarrow |M_{uv}|$ ; if  $d < n$  then  $M_u \leftarrow M_{uv}$ ;  $M_v \leftarrow \varepsilon$ 
14 else  $M_u \leftarrow M_{uv}[1..n]$ ;  $M_v \leftarrow M_{uv}[n+1..|M_{uv}|]$  fi
15 for  $i \leftarrow 1$  to  $m$  do  $W_i \leftarrow M_i \oplus f_{a,i}(M'_i)$ ;  $X_i \leftarrow M'_i \oplus f_a(W_i)$  od
16 if  $d = 0$  then  $X \leftarrow X_1 \oplus \dots \oplus X_m \oplus \mathbf{0}$ 
17 else if  $d < n$  then  $X \leftarrow X_1 \oplus \dots \oplus X_m \oplus f_u(M_u 10^*)$ 
18 else  $X \leftarrow X_1 \oplus \dots \oplus X_m \oplus f_u(M_u) \oplus f_v(M_v 10^*)$  fi
19  $S_x \leftarrow M_x \oplus X \oplus f_x(M_y)$ ;  $S_y \leftarrow M_y \oplus f_{xx}(S_x)$ ;  $S \leftarrow S_x \oplus S_y$ 
20 for  $i \leftarrow 1$  to  $m$  do
21      $S' \leftarrow f_{aa,i}(S)$ ;  $Y_i \leftarrow W_i \oplus S'$ ;  $Z_i \leftarrow X_i \oplus S'$ 
22      $C'_i \leftarrow Y_i \oplus f_a(Z_i)$ ;  $C_i \leftarrow Z_i \oplus f_{a,i}(C'_i)$  od
23 if  $d = 0$  then  $C_u \leftarrow C_v \leftarrow \varepsilon$ ;  $Y \leftarrow Y_1 \oplus \dots \oplus Y_m \oplus \mathbf{0}$ 
24 else if  $d < n$  then  $C_u \leftarrow M_u \oplus f_{uu}(S)$ ;  $C_v \leftarrow \varepsilon$ ;  $Y \leftarrow Y_1 \oplus \dots \oplus Y_m \oplus f_u(C_u 10^*)$ 
25 else  $C_u \leftarrow M_u \oplus f_u(S)$ ;  $C_v \leftarrow M_v \oplus f_v(S)$ 
26      $Y \leftarrow Y_1 \oplus \dots \oplus Y_m \oplus f_u(C_u) \oplus f_v(C_v 10^*)$  fi
27  $C_y \leftarrow S_x \oplus f_{yy}(S_y)$ ;  $C_x \leftarrow S_y \oplus Y \oplus f_y(C_y)$ 
28 return  $C_1C'_1 \dots C_mC'_m C_uC_v C_xC_y$ 

```

Fig. 4. The AEZ-core[f] construction. The method builds a strong-PRP on $\{0, 1\}^{\geq 2n}$ from an n -bit-output PRF f that operates on its subscript and argument. Its key K is implicit. The PRF can be realized by a TBC.

When AEZ-core is turned into an RAE scheme by the EtE construction, this observation is put to good use in achieving fast rejection of ciphertexts whose final 0^τ bits is plaintext is not correct. That it is undamaging to release this timing information is guaranteed by results already show—in particular, that it is ok to release the *entire* speculative plaintext.

AEZ-core confines “specialized” processing to the final 2–4 blocks. This helps with efficiency and simplicity compared to having specialized processing at the beginning or at the beginning and end. In particular, the 0^τ authenticator used to make an RAE scheme will be put at the *end* of the message (adding a variable number of zero-bits at beginning could destroy word alignment) and, as long as $\tau \leq 2n$, it will be found in the final two blocks.

Numerous alternatives to AEZ-core were considered before arriving at our design. Correct alternatives we know are slower or more complex, while most simplifications are wrong. For example, consider trying to cheapen the design by using $c_i \cdot f_{\text{aa},1}(S)$ instead of $f_{\text{aa},i}(S)$ where each c_i is a public constant and the product is in $\text{GF}(2^n)$. This fails for *any* choice of c_i . See Appendix C.

One variant of AEZ-core that *does* work is to eliminate the “left-hand” xor coming out of $f_{\text{aa},i}$. (One then has to define X_i as the output of f_a instead of that output xor’ed with M'_1 , and change Y_i similarly.) We have kept this xor because it’s needed for symmetry.

7 Definition of AEZ

So far we have described two key elements of AEZ: the EtE construction and the AEZ-core[f] wide-block blockcipher. Now we give AEZ’s complete description. First a bit of notation.

NOTATION. The bit length of a string X is written $|X|$. For the bitwise xor of unequal-length strings, drop the necessary number of rightmost bits from the longer ($10 \oplus 0100 = 11$). For X a string, let $X0^* = X0^p$ with p the smallest number such that 128 divides $|X| + p$. By \mathcal{X}^* we denote the set of all strings over the alphabet \mathcal{X} , including ε . By $(\mathcal{X}^*)^*$ we denote the set of all vectors over \mathcal{X}^* , including the empty vector.

If $|X| = n$ and $1 \leq i \leq j \leq n$ then $X(i)$ is the i th bit of X (indexing from the left starting at 1), $\text{msb}(X) = X(1)$, and $X(i..j) = X(i) \cdots X(j)$. Let $[n]_t$ be the t -bit string representing $n \bmod 2^t$ and let $[n]$ be shorthand for $[n]_8$; for example $[0]^{16} = ([0]_8)^{16} = 0^{128}$ and $[1]^{16} = (00000001)^{16}$. A block is 128 bits. Let $\mathbf{0} = 0^{128}$. If $X = a_1 \cdots a_{128}$ is a block ($a_i \in \{0, 1\}$) then we define $X \ll 1 = a_2 \cdots a_{128} 0$. For $n \in \mathbb{N}$ and $X \in \{0, 1\}^{128}$ define $n \cdot X$ by asserting that $0 \cdot X = \mathbf{0}$ and $1 \cdot X = X$ and $2 \cdot X = (X \ll 1) \oplus [135 \cdot \text{msb}(X)]_{128}$ and $2n \cdot X = 2 \cdot (n \cdot X)$ and $(2n + 1) \cdot X = (2n \cdot X) \oplus X$.

For $K, X \in \{0, 1\}^{128}$ we write $\text{aesenc}(X, K)$ for a single round of AES: SubBytes, ShiftRows, MixColumns, then an AddRoundKey with K . For $\mathbf{K} = (K_0, K_1, K_2, K_3, K_4)$ a list of five blocks, let $\text{AES4}_{\mathbf{K}}(X) = \text{AES4}(\mathbf{K}, X)$ be $\text{aesenc}(\text{aesenc}(\text{aesenc}(X \oplus K_0, K_1), K_2), K_3), K_4)$. For \mathbf{K} a list of 11 blocks, $\mathbf{K} = (K_0, K_1, \dots, K_{10})$, define $\text{AES10}_{\mathbf{K}}(X) = \text{AES10}(\mathbf{K}, X)$ like we defined AES4 but with ten rounds of aesenc . We do not omit the final-round MixColumns.

AEZ DEFINITION. See Figs. 5 and 6 for the definition of AEZ, and Fig. 7 for an illustration. Most of it is self-explanatory. We briefly explain some of the algorithm’s more unusual elements.

AEZ operates on arbitrary byte strings. Not only is the plaintext $M \in \text{BYTE}^*$ arbitrary, but so too the key $\text{Key} \in \text{BYTE}^*$ and nonce $N \in \text{BYTE}^*$. The AD is even more general: an arbitrary-length vector of byte strings, $A \in (\text{BYTE}^*)^*$. The requested ciphertext expansion of $\lambda \in \mathbb{N}$ bytes is measured in $\tau = 8\lambda$ bits.

```

100 algorithm ENCRYPT( $K, N, A, \tau, M$ ) //AEZ authenticated encryption
101  $X \leftarrow M \parallel 0^\tau$ ;  $(A_1, \dots, A_m) \leftarrow A$ 
102  $T \leftarrow ([\tau]_{128}, N, A_1, \dots, A_m)$ 
103 if  $M = \varepsilon$  then return AEZ-prf( $K, T, \tau$ ) else return Encipher( $K, T, X$ )
110 algorithm DECRYPT( $K, N, A, \tau, C$ ) //AEZ authenticated decryption
111  $(A_1, \dots, A_m) \leftarrow A$ ;  $T \leftarrow ([\tau]_{128}, N, A_1, \dots, A_m)$ 
112 if  $|C| < \tau$  then return  $\perp$ 
113 if  $|C| = \tau$  then if  $C = \text{AEZ-prf}(K, T, \tau)$  then return  $\varepsilon$  else return  $\perp$  fi fi
114  $X \leftarrow \text{Decipher}(K, T, C)$ ;  $M \parallel Z \leftarrow X$  where  $|Z| = \tau$ 
115 if  $(Z = 0^\tau)$  then return  $M$  else return  $\perp$ 
200 algorithm Encipher( $K, T, X$ ) //AEZ enciphering
201 if  $|X| < 256$  then return Encipher-AEZ-tiny( $K, T, X$ )
202 if  $|X| \geq 256$  then return Encipher-AEZ-core( $K, T, X$ )
210 algorithm Encipher-AEZ-tiny( $K, T, M$ ) //AEZ-tiny enciphering
211  $m \leftarrow |M|$ ;  $n \leftarrow m/2$ ;  $\Delta \leftarrow \text{AEZ-hash}(K, T)$ 
212 if  $m = 8$  then  $k \leftarrow 24$  else if  $m = 16$  then  $k \leftarrow 16$ 
213 else if  $m < 128$  then  $k \leftarrow 10$  else  $k \leftarrow 8$  fi
214  $L \leftarrow M(1..n)$ ;  $R \leftarrow M(n+1..m)$ ; if  $m \geq 128$  then  $j \leftarrow 6$  else  $j \leftarrow 7$  fi
215 for  $i \leftarrow 0$  to  $k-1$  do
216    $R' \leftarrow L \oplus ((E_K^{0,j}(\Delta \oplus R10^* \oplus [i]_{128}))(1..n))$ ;  $L \leftarrow R$ ;  $R \leftarrow R'$  od
217  $C \leftarrow R \parallel L$ ; if  $m < 128$  then  $C \leftarrow C \oplus (E_K^{0,3}(\Delta \oplus (C0^* \vee 10^*)) \wedge 10^*)$  fi
218 return  $C$ 
220 algorithm Encipher-AEZ-core( $K, T, M$ ) //AEZ-core enciphering
221  $M_1 M'_1 \dots M_m M'_m M_{uv} M_x M_y \leftarrow M$ 
222   where  $|M_1| = \dots = |M'_m| = |M_x| = |M_y| = 128$  and  $|M_{uv}| < 256$ 
223  $\Delta \leftarrow \text{AEZ-hash}(K, T)$ ;  $d \leftarrow |M_{uv}|$ 
224 if  $d \leq 127$  then  $M_u \leftarrow M_{uv}$ ;  $M_v \leftarrow \varepsilon$ 
225 else  $M_u \leftarrow M_{uv}[1..128]$ ;  $M_v \leftarrow M_{uv}[129..|M_{uv}|]$  fi
226 for  $i \leftarrow 1$  to  $m$  do  $W_i \leftarrow M_i \oplus E_K^{1,i}(M'_i)$ ;  $X_i \leftarrow M'_i \oplus E_K^{0,0}(W_i)$  od
227 if  $d = 0$  then  $X \leftarrow X_1 \oplus \dots \oplus X_m \oplus \mathbf{0}$ 
228 else if  $d \leq 127$  then  $X \leftarrow X_1 \oplus \dots \oplus X_m \oplus E_K^{0,4}(M_u 10^*)$ 
229 else  $X \leftarrow X_1 \oplus \dots \oplus X_m \oplus E_K^{0,4}(M_u) \oplus E_K^{0,5}(M_v 10^*)$  fi
230  $S_x \leftarrow M_x \oplus \Delta \oplus X \oplus E_K^{0,1}(M_y)$ ;  $S_y \leftarrow M_y \oplus E_K^{-1,1}(S_x)$ ;  $S \leftarrow S_x \oplus S_y$ 
231 for  $i \leftarrow 1$  to  $m$  do
232    $S' \leftarrow E_K^{2,i}(S)$ ;  $Y_i \leftarrow W_i \oplus S'$ ;  $Z_i \leftarrow X_i \oplus S'$ 
233    $C'_i \leftarrow Y_i \oplus E_K^{0,0}(Z_i)$ ;  $C_i \leftarrow Z_i \oplus E_K^{1,i}(C'_i)$  od
234 if  $d = 0$  then  $C_u \leftarrow C_v \leftarrow \varepsilon$ ;  $Y \leftarrow Y_1 \oplus \dots \oplus Y_m \oplus \mathbf{0}$ 
235 else if  $d \leq 127$  then
236    $C_u \leftarrow M_u \oplus E_K^{-1,4}(S)$ ;  $C_v \leftarrow \varepsilon$ ;  $Y \leftarrow Y_1 \oplus \dots \oplus Y_m \oplus E_K^{0,4}(C_u 10^*)$ 
237 else  $C_u \leftarrow M_u \oplus E_K^{-1,4}(S)$ ;  $C_v \leftarrow M_v \oplus E_K^{-1,5}(S)$ 
238    $Y \leftarrow Y_1 \oplus \dots \oplus Y_m \oplus E_K^{0,4}(C_u) \oplus E_K^{0,5}(C_v 10^*)$  fi
239  $C_y \leftarrow S_x \oplus E_K^{-1,2}(S_y)$ ;  $C_x \leftarrow S_y \oplus \Delta \oplus Y \oplus E_K^{0,2}(C_y)$ 
240 return  $C_1 C'_1 \dots C_m C'_m C_u C_v C_x C_y$ 

```

Fig. 5. Main routines of AEZ. The tweakable blockcipher E , the hash AEZ-hash, and the PRF AEZ-prf are defined in Fig. 6. The ciphertext expansion is $\tau = 8\lambda$ bits.

```

300 algorithm AEZ-hash( $K, T$ ) //AXU hash.  $T$  is a vector of strings
301  $(T_1, \dots, T_t) \leftarrow T$ 
302 for  $i \leftarrow 1$  to  $t$  do
303    $m \leftarrow \max(1, \lceil |T_i|/128 \rceil)$ ;  $X_1 \cdots X_m \leftarrow T_i$  // $|X_1| = \dots = |X_{m-1}| = 128$ 
304   if  $|X_m| = 128$  then  $\Delta_i \leftarrow E_K^{2+i,1}(X_1) \oplus \dots \oplus E_K^{2+i,m}(X_m)$ 
305   if  $|X_m| < 128$  then
306      $\Delta_i \leftarrow E_K^{2+i,1}(X_1) \oplus \dots \oplus E_K^{2+i,m-1}(X_{m-1}) \oplus E_K^{2+i,0}(X_m 10^*)$ 
307 return  $\Delta_1 \oplus \dots \oplus \Delta_t \oplus \mathbf{0}$ 

310 algorithm AEZ-prf( $K, T, \tau$ ) //PRF used when  $M = \varepsilon$ 
311  $\Delta \leftarrow \text{AEZ-hash}(K, T)$ 
312 return  $(E_K^{-1,3}(\Delta) \parallel E_K^{-1,3}(\Delta \oplus [1]_{128}) \parallel E_K^{-1,3}(\Delta \oplus [2]_{128}) \parallel \dots)[1..\tau]$ 

400 algorithm  $E_K^{i,j}(X)$  //Scaled-down TBC
401  $I \parallel J \parallel L \leftarrow \text{Extract}(K)$  where  $|I| = |J| = |L| = 128$ 
402  $k_0 \leftarrow (\mathbf{0}, I, J, L, \mathbf{0}); k_1 \leftarrow (\mathbf{0}, J, L, I, \mathbf{0}); k_2 \leftarrow (\mathbf{0}, L, I, J, I)$ 
403  $K \leftarrow (\mathbf{0}, I, L, J, I, L, J, I, L, J, I)$ 
404 if  $i = -1$  and  $0 \leq j \leq 7$  then return AES10 $_K(X \oplus jJ)$ 
405 if  $i = 0$  and  $0 \leq j \leq 7$  then return AES4 $_{k_0}(X \oplus jJ)$ 
406 if  $1 \leq i \leq 2$  and  $j \geq 1$  then return AES4 $_{k_i}(X \oplus (j \bmod 8)J \oplus 2^{\lfloor(j-1)/8\rfloor}L)$ 
407 if  $i \geq 3$  and  $j \geq 1$  then
408   return AES4 $_{k_0}(X \oplus (j \bmod 8)J \oplus 2^{\lfloor(j-1)/8\rfloor} \cdot L \oplus (i-2)8J)$ 
409 if  $i \geq 3$  and  $j = 0$  then return AES4 $_{k_0}(X \oplus (i-2)8J)$ 

```

Fig. 6. AEZ’s hash, PRF, and TBC. The last is the locus of prove-then-prune scaling-down. The key K is turned into 384 bits by a key-derivation function Extract.

At line 217, Encipher-AEZ-tiny may xor a bit into the ciphertext just before the algorithm’s conclusion. This is done to avoid a simple random-permutation distinguishing attacks, for very short strings, based on the fact that Feistel networks only generate *even* permutations [30]. A similar trick, conditionally swapping two fixed points, has been used before [45]. Our approach has the benefit that the natural implementation is constant-time.

We define Decipher(K, T, Y) as the unique X such that Encipher(K, T, X) = Y . Logically, this is all we need say for the specification to be well-defined. Still, the additional pseudocode is easy to describe. AEZ-tiny deciphering is identical to AEZ-tiny enciphering except we must count backwards instead of forwards, and must do the even-cycles correction (line 217) at the beginning instead of the end. Specifically, Decipher-AEZ-tiny(K, T, M) is identical to Encipher-AEZ-tiny(K, T, M) except that line 215 is changed to count from $k - 1$ down to 0, while line 217 has each C replaced by M before moving the line up to just after line 213. And AEZ-core deciphering is identical to AEZ-core enciphering except that we must take the xy -tweaks in reverse order. Specifically, Decipher-AEZ-core(K, T, M) is identical to Encipher-AEZ-core(K, T, M) except we swap tweaks $(0, 1)$ and $(0, 2)$, and we swap tweaks $(-1, 1)$ and $(-1, 2)$. These appear at lines 230 and 239.

The TBC $E_K^{i,j}(X)$ takes a tweak $(i, j) \in \{-1, 0\} \times [0..7] \cup \{1, 2, 3\} \times \mathbb{N}$. The first component selects between AES10 (when $i = -1$) and AES4 (when $i \geq 0$).

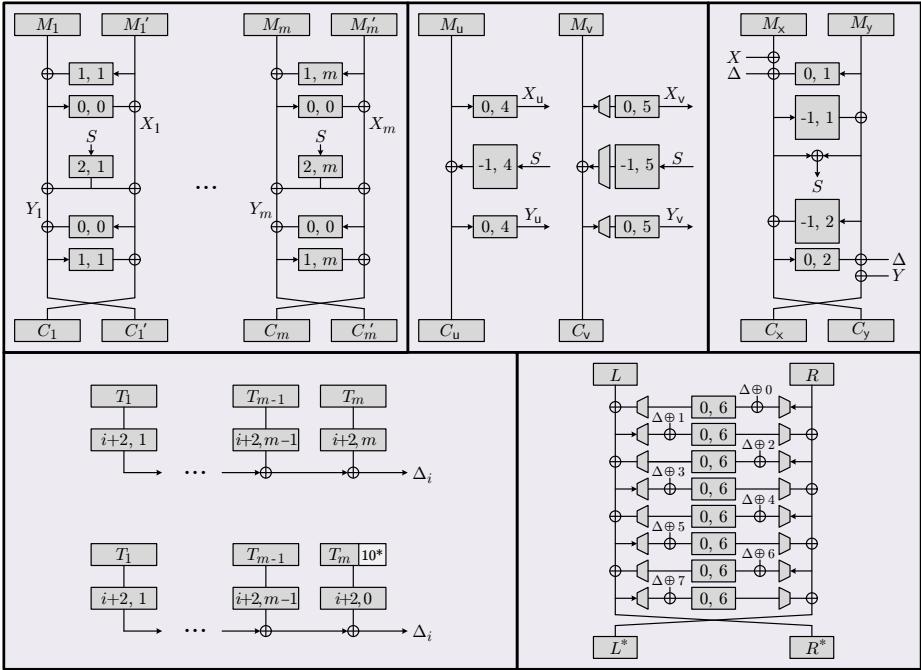


Fig. 7. Illustrating AEZ enciphering. Rectangles with pairs of numbers are TBCs, the pair being the tweak (the key, always K , is not shown). **Top row:** enciphering a message M of (32 or more bytes) with AEZ-core. The diagram shows processing a string that is (exclude the middle panel) or isn't (include the middle panel) a multiple of 16 bytes. **Bottom left:** AEZ-hash is an xor-universal hash built from AES4. It computes $\Delta = \bigoplus \Delta_i$ from a vector-valued tweak T comprising A , N , and τ . Its i -th component $T_1 \dots T_m$ is hashed as shown. **Bottom right:** AEZ-tiny, when operating on a string $M = L \parallel R$ of 16–31 bytes. More rounds are used if M has 1–15 bytes.

Either way, the construction is based on XE [34, 49]. We emphasize that E is *not* secure as a tweakable-PRP, since AES4 itself is completely insecure as a PRP: it is easily broken by the “Square” attack [14]. Use of an AES4-based TBC despite this fact is where the scaling-down has been done in AEZ.

The key $K \in \text{BYTE}^*$ is mapped to three 16-byte subkeys (I, J, L) using the key-derivation function (KDF) named Extract that is called at line 401. The definition of Extract is omitted from the figures and regarded as orthogonal to the rest of AEZ. See the AEZ spec [26] for the current Extract: $\text{BYTE}^* \rightarrow \text{BYTE}^{48}$. In our view, it is an unresolved matter what the security properties (and even what signature) of a good KDF should be. Work has gone off in very different directions [33, 46, 61], and the area is currently the subject of a Password Hashing Competition (PHC) running concurrently with CAESAR.

Note the mod 8’s at lines 406 and 408. Unlike the offset sequence used for OCB [32], we limit ourselves to eight successive J values; after that, we add in the next power-of-two times L . This allows a small table of $2^j \cdot J$ values to be

precomputed and used regardless of the length of the message. In this way we limit the frequency of doublings yet avoid number-of-trailing-zeros calculations.

We impose a limit that AEZ be used for at most 2^{48} bytes of data (about 280 TB); by that time, the user should rekey. This usage limit stems from the existence of birthday attacks on AEZ, as well as the use of AES4 to create a universal hash function.

COST ACCOUNTING. Let us summarize the computational cost of AEZ in “AES-equivalents,” where 1 AES-equivalent is 10 AES rounds. Assume a message of m blocks, the last of which may be fragmentary. To encipher or decipher $m \geq 2$ blocks takes at most $m + 2.4$ AES-equivalents (latency 3.6). This assumes K , N , τ , and A have already been processed. To encrypt or decrypt $m \geq 2$ blocks: at most $m + 3.8$ AES-equivalents (latency 3.6). This assumes that K , A , and τ have already been processed and that $|N| \leq 128$ and $\tau = 128$. To reject an invalid ciphertext of $m \geq 2$ blocks: at most $0.4m + 2.4$ AES-equivalents (latency 2.8). Same assumptions. To setup an m block key: $1.2m$ AES-equivalents (latency 0.4). This assumes that needed constants have been precomputed. To setup a string-values AD: $0.4m$ (latency 0.4). To encipher or decipher messages of 1–15 bytes is somewhat slower: 10, 6.8, and 4.4 AES-equivalents for 1, 2, and 3 bytes.

PARAMETERIZED COUNTERPARTS. For a TBC-parameterized generalization of AEZ, let $\text{AEZ}[\tilde{E}]$ be identical to AEZ except for using the TBC $\tilde{E}: \mathcal{K} \times \mathcal{T}_{\text{aez}} \times \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ in place of E (assume the correct tweak-space \mathcal{T}_{aez}). The key space of \tilde{E} is then taken as the key space for the constructed RAE scheme. Note that $\text{AEZ} = \text{AEZ}[E]$ with E the algorithm defined by lines 400–409.

Taking the above a step further, given a conventional blockcipher $E: \mathcal{K} \times \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ we can define $\text{AEZ}[E]$ as $\text{AEZ}[\tilde{E}]$ where $\tilde{E}_K^{i,j}(X) = E_K(X \oplus (i+1)I \oplus jJ)$ for $I = E_K(\mathbf{0})$ and $J = E_K(\mathbf{1})$. The scheme $\text{AEZ}[\text{AES}]$ can be regarded as a natural “scaled up” version of AEZ. We emphasize that AEZ is *not* $\text{AEZ}[\text{AES}]$, which is about 2.5 times as slow.

Schemes $\text{AEZ}[\tilde{E}]$ and $\text{AEZ}[E]$ are close to AEZ, but enjoy conventional provable-security guarantees, as we now describe.

8 Security of $\text{AEZ}[\tilde{E}]$ and $\text{AEZ}[E]$

We show that if \tilde{E} is secure as a tweakable PRP then $\text{AEZ}[\tilde{E}]$ is RAE-secure. In fact, the statement holds even if the decryption algorithm is modified so as to leak the entire improperly encoded string obtained by deciphering an invalid ciphertext. So, for the remainder of this section, assume the modification of AEZ in which the **else** clause of line 115 returns the deciphered message X rather than \perp . This change only makes our results stronger, explicitly modeling the *possibility* of a decryption implementation leaking some or all of X . The *actual* decryption algorithm returns \perp .

Our provable-security results for AEZ need to assume that the adversary avoids enciphering or deciphering extremely short strings—at least those under

16 bytes, say, for which AEZ-tiny, a Feistel-based construction, will not enjoy a desirable bound. While provably-secure options are now available for enciphering very short strings, they still do not have competitive efficiency.

As the alphabet for AEZ is $\Sigma = \text{BYTE}$, in this section we write $|x|$ for the byte length of x . For an encryption query (N, A, λ, M) , define the number of blocks processed as $\lceil |N|/16 \rceil + \sum_i \lceil |A_i|/16 \rceil + \lceil (|M| + \lambda)/16 \rceil$. This query is *small* if $M \neq \varepsilon$ and $16 \leq |M| + \lambda < 32$, and *tiny* if $M \neq \varepsilon$ and $|M| + \lambda < 16$. Likewise, for a decryption query (N, A, λ, C) , the number of blocks processed is $\lceil |N|/16 \rceil + \sum_i \lceil |A_i| \rceil + \lceil (|C|)/16 \rceil$. The query is *small* if $16 \leq |C| < 32$ and $|C| \neq \lambda$, and *tiny* if $|C| \neq \lambda$ and $|C| < 16$. The proof for the following is in the full version [28].

Theorem 5. Let $\tilde{E} : \mathcal{K} \times \mathcal{T}_{\text{aez}} \times \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ be a TBC and $\Pi = \text{AEZ}[\tilde{E}]$. Then there are efficient, explicitly given algorithms \mathcal{R} and \mathcal{S} with the following property. Let \mathcal{A} be an adversary for attacking Π . Assume it never asks any small or tiny query. Then $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}_{\Pi,S}^{\text{rae}}(\mathcal{A}) \leq 3.5s^2/2^{128} + \mathbf{Adv}_{\tilde{E}}^{\text{PPF}}(\mathcal{B})$, where s is the total number of processed blocks, plus 2 blocks per message. Adversary \mathcal{B} makes at most $2.5s$ queries and has about the same running time as \mathcal{A} .

An alternative approach to justifying the security of AEZ is to speak of the security of $\text{AEZ}[E]$, the cousin of AEZ defined from a conventional blockcipher E using the XE construction to make the needed TBC. Its security can be captured by the following result. The proof is in the full version [28].

Theorem 6. Let $E : \mathcal{K} \times \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ be a blockcipher and $\Pi = \text{AEZ}[E]$. Then there are efficient, explicitly given algorithms \mathcal{R} and \mathcal{S} with the following property. Let \mathcal{A} be an adversary for attacking Π . Assume it never asks a small or tiny query. Then $\mathcal{B} = \mathcal{R}(\mathcal{A})$ satisfies $\mathbf{Adv}_{\Pi,S}^{\text{rae}}(\mathcal{A}) \leq 13s^2/2^{128} + \mathbf{Adv}_E^{\text{PPF}}(\mathcal{B})$, where s is the total number of processed blocks, plus 2 blocks per message. Adversary \mathcal{B} makes at most $2.5s$ queries and has about the same running time as \mathcal{A} .

If one wants to accommodate small queries then we still have a provable, albeit much inferior result. Let $\text{Feistel}[r, n]$ denote an ideal r -round Feistel network on $\{0,1\}^{2n}$. The best known provable bound for Feistel networks [43, Theorem 7] states that if an adversary makes $q \leq \frac{2^n}{128n}$ queries then $\mathbf{Adv}_{\text{Feistel}[6,n]}^{\pm\text{PPF}}(\mathcal{A}) \leq \frac{8q}{2^n} + \frac{q^2}{2^{2n+1}}$. Translating this to our setting, one is bound to make at most $q \leq \frac{2^{64}}{128 \cdot 64} = 2^{51}$ small queries, and the security advantage is $q/2^{61} + 4s^2/2^{128}$. These restrictions seem to be more of the artifacts of the analysis in [43, Theorem 7] than reflecting the actual security of Feistel networks: assuming that the round functions of $\text{Feistel}[6, n]$ are instantiated from full AES, the fastest known attack, for $n \geq 64$, is still the exhaustive key search on AES.

9 Estimated Security of AEZ Itself

Consider enciphering a message M , $|M| \geq 256$, by AEZ[AES] (which, recall, is not AEZ, but a scaled-up version using an AES-based TBC). The design would seem excessive: each block M_i would be subjected to 30 rounds of AES (ten shared with a neighboring block), not counting the additional AES rounds to produce the highly unpredictable, M -dependent value S , a value derived from which gets injected into the process while 20 rounds yet remain. It is in light of such apparent overkill that AEZ selectively prunes some of the AES calls that AEZ[AES] would perform. In particular, we prune invocations where we aim to achieve computational xor-universal hashing. We leave enough AES rounds so that each block M_i is effectively processed with 12 AES rounds, eight of these subsequent to injection of the highly-unpredictable S and four of them shared with a neighboring block. The key steps in calculating S are not pruned, nor are the TBCs used to mask u- and v-blocks.

To estimate the security of AEZ it seems appropriate to replace the $s^2/2^{128}$ term of Theorem 5 by $s^2/2^{113}$, resulting in the bound $4s^2/2^{113} + t/2^{128}$, because of the higher maximal expected differential probability of AES4 [31] compared to an ideal hash or cipher, where t is the time (including the description size) in which the adversary runs.

Moreover, we contend that the assumption that the adversary avoids asking tiny or small queries can be lifted. To justify this heuristically, consider a collection of independent, ideal, k -round Feistel networks on $\{0, 1\}^{2n}$; the round functions are all uniformly random and independent. The best attack known, due to Patarin [41], that distinguishes them from a family of independent, truly random even permutations requires at least $2^{(k-4)n}$ plaintext/ciphertext pairs. From our choice of the number of rounds, this attack needs at least 2^{72} plaintext/ciphertext pairs, and thus doesn't violate our up-to-the-birthday-bound security goal.

AEZ was specifically designed so that scaling-down most of its AES calls would seem safe. This is design-specific; one cannot indiscriminately scale a scheme's primitives. A previous design, where AEZ-core followed the NR approach [39, 40], could not be as effectively scaled-down.

10 Software Performance

The development of AEZ has generally presumed an instruction set architecture (ISA) with round-level support for AES, such as Intel's AES-NI or ARM's version 8 ISA. On these systems the AES unit can be kept busy processing several AES4 computations in parallel while idle processing units handle load, store, and xor overhead. On Intel's Haswell architecture, for example, unrelated AES rounds can issue every cycle and take seven cycles to retire, so seven parallel AES4 calculations can complete in 34 CPU cycles, while idle superscalar processing units can handle other computations. This observation has led us to design AEZ to conveniently process eight blocks at a time.

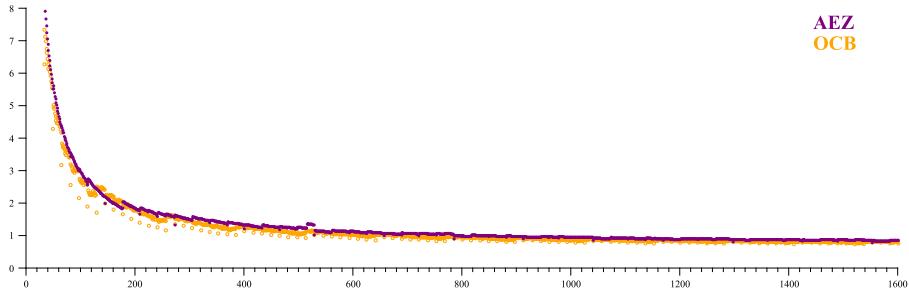


Fig. 8. AEZ vs. OCB performance. The x -axis is message length, in bytes, and the y -axis is cycles per byte (cpb). The graph is best viewed in color: solid purple circles are for AEZ; unfilled yellow circles are for OCB3 [32]. Performance of the two is close, both having peak speeds around 0.7 cpb and being similar on most shorter messages as well. The execution vehicle is an Intel Haswell processor using AES-NI.

AEZ overhead beyond AES rounds has been minimized. As an example of this, our AES4 key schedule omits the final round key, allowing `aesenc`'s included xor operation to be used for other purposes. Such optimizations lead to AEZ peak speeds, on Haswell, of around 0.72 cpb—not far from the theoretical maximum for the number of rounds executed of 0.63 cpb.

Fig. 8 compares the performance of AEZ and OCB on messages of all byte lengths up to 1600 bytes. The two are not only similar for long messages but for short strings too. Only when messages are shorter than 16 bytes, where AEZ-tiny increases the number of AES4 calls used, does OCB become significantly faster.

The performance of AEZ is on par with OCB even on processors that are not superscalar or do not support AES rounds at the assembly level. On a Marvell 88F6283 embedded CPU—a single-issue, 32-bit, ARM version 5 ISA—we see an experimental version of AEZ peaking at 86 cpb while OCB's optimized reference code runs at 84 cpb. For comparison, GCM, CCM and CTR run at 124, 134 and 67 cpb, respectively. The figures use the OpenSSL libraries.

One might expect the two-pass nature of AEZ to be a performance burden because data must be dragged into cache from memory twice. We have found that modern processors, like Intel's Haswell, have such efficient hardware prefetching that bringing data into cache twice, in a sequential streaming fashion, is not expensive at all. It requires no explicit prefetching. Encrypting 1MB on Haswell is as efficient as encrypting 32KB despite 1MB exceeding the 256KB level-2 cache. Two passes may have a more significant cost on systems with poor prefetching facilities, although this might be mitigated by software prefetching.

Another benefit of AEZ's two passes is that the second pass is not needed to discover that a ciphertext is inauthentic, leading to message rejection costing as little as 0.28 cpb on Haswell. On long messages, approximately 2/5 of AES4 calls are performed during the first pass, which aligns perfectly with the peak times we've observed for encryption and fast-rejection.

All Haswell timings reported in this paper were gathered on a 2.9 GHz Intel Core i5-4570S CPU using its time-stamp counter to gather elapsed CPU cycles

over encryption calls. Our implementation is written in C using “intrinsic” functions to access CPU-specific functionality. It was compiled using GCC 4.9 with options `-march=native -O3`. Our optimized implementation will be made publicly available and freely licensed.

Acknowledgments. Many thanks to Tom Shrimpton, who provided important interaction on RAE definitions and their implications. Liden Mu and Chris Patton proofread our specification document and did implementations that helped verify our own. We received good comments and corrections Danilo Gligoroski, Tom Ristenpart, and Yusi (James) Zhang. Thanks to Dustin Boswell for an April 2013 email on the importance of making AE easier to use, Stefan Lucks for a Jan 2012 discussion on the problem unverified plaintexts, and René Struik for an August 2013 DIAC presentation on the utility of minimizing ciphertext expansion. Thanks to Terence Spies for catalyzing the idea of unifying AE and blockciphers both in definition and schemes.

Part of this work was done when Tung was a postdoc at UC San Diego and Phil was visiting ETH Zürich. Many thanks to Mihir Bellare for that postdoc, and many thanks to Ueli Maurer for hosting that sabbatical.

Hoang was supported by NSF grants CNS-0904380, CCF-0915675, CNS-1116800 and CNS-1228890; Krovetz was supported by NSF grant CNS-1314592; and Rogaway was supported by NSF grants CNS-1228828 and CNS-1314885. Many thanks to the NSF for their continuing support.

A More on Related Work

RAE and AEZ build on a large body of related work. While we have summarized much of this throughout this paper, here we give some additional context and high points.

Blockciphers accommodating truly arbitrary inputs were first realized by Schroepell’s Hasty Pudding Cipher (HPC) [55]. Ahead of its time, the work not only built a blockcipher on all of $\{0, 1\}^*$, but also provided it a tweak. If one were to first overcome the problem that HPC’s tweak is limited in length, it could be used with the EtE construction to make an RAE scheme.

The problem of constructing from conventional blockciphers those with arbitrary or near-arbitrary domains was first identified Bellare and Rogaway [4], who wanted to construct these objects with a conventional-looking mode. But the mechanism they suggested was somewhat slow, was limited to a domain of $(\{0, 1\}^n)^+$, and only achieves conventional (not strong) PRP security.

In a follow-up paper [5] the same authors evidenced the utility of arbitrary-input-length blockciphers by explaining how semantic security could be achieved by enciphering messages with novelty, and they showed how authenticity could be achieved by enciphering messages with redundancy (this time using a *strong* PRP). These observations formed the basis for our work.

Around the same time as the last two work, Naor and Reingold (NR) constructed a blockcipher on $(\{0, 1\}^n)^+$ by sandwiching a layer of ECB between layers of a “blockwise-universal” hashing [39, 40]. The approach came to be used in many proposals, including XCB [35], which was standardized in the IEEE [29].

The other method inspiring further wide-block blockciphers was EME [24], which involves two layers of blockcipher-based enciphering and a light layer of mixing in between. A follow-on design, EME2 [22], became the other wide-block blockcipher of IEEE 1619.2 [29]. Both it and XCB are tweakable and operate on a message space of $\{0,1\}^{\geq n}$. EME/EME2 provides the starting point for AEZ-core.

As for extending blockciphers to short blocks, a different line of work was begun [9]. *Format-preserving encryption* aimed to deal not only with small domains but also those defined as arbitrary finite sets, sets of numbers $[0..N-1]$, or strings over arbitrary alphabets. Adapting Feistel designs to arbitrary alphabets, realizations of FFX [6], now a draft NIST standard [17], would form the basis of AEZ-tiny.

Meanwhile, notions of AE were appearing. Probabilistic versions came first [5, 27], then a nonce-based version [50], then AD finally appeared [49]. Next the MRAE goal—RAE’s closest definition counterpart—was defined [51]. The main motivation for that work was to minimize the damage that could be done by nonce-reuse.

Other authors had the same concern but weren’t willing to use two-pass schemes. Fleischmann *et. al* [20] built on Bellare *et. al* [3] to define a security notion for online-AE intended to confer some lower level of nonce-reuse misuse-resistance. The approach has gained popularity—many CAESAR submissions follow it, especially after COPA [2] made clear that one could achieve this weakened flavor of nonce-reuse misuse-resistance with a parallelizable scheme. The RAE definition goes a different direction, strengthening instead of weakening the original MRAE definition.

Following up on directions from prior work [10, 20, 21], AE security in the face of decryption-algorithm leakage was studied by Andreeva *et. al* [1] in work concurrent with our own. A principle motivation for those authors has been to express when it is OK for an online decryption algorithm to be incrementally releasing unverified plaintext. For us, this is a direction not taken, for such leakage can never be generically harmless [47]. In effect, leaking equality of message prefixes is leaking an enormous amount of information.

Ferguson made clear early on that AE algorithms could fail badly when tags are too short [18]. Still, no definitions for AE security were ever offered appropriate to the short-tag setting. But the general concern for making short MACs work well goes back to Black and Cochran [8] and Wang *et. al* [59].

Some examples of using AES4 where AES itself would do include ALRED, LETTERSOUP, MARVIN, and Pelican [15, 16, 57]. These inspired our predilection to cut certain AES rounds even when provable security couldn’t promise this was fine. The approach should not be confused with that of Minematsu and Tsunoo [37], where AES4 provably *does* suffice for the protocol devised [37]. The approach leverages the low MEDP for AES4, a line of work culminating in the bound of Keliher and Sui [31].

Many authors have proposed ideas to eliminate use of the inverse-direction of a blockcipher in modes that previously needed this. The method we use in AEZ is inspired by Minematsu's OTR [36].

The CAESAR competition [7], organized by Dan Bernstein, was the proximal motivation to define RAE and to try to develop a nice scheme for achieving it.

B Deferred Proofs

B.1 Proof of Theorem 1

It suffices to show that

$$|\Pr[\mathcal{A}^{\mathbf{Ideal}_H} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{PRI}_H} \Rightarrow 1]| \leq (r^2 + r)/|\Sigma|^{\lambda+m_{\min}+1} + 2q/|\Sigma|^\lambda.$$

Without loss of generality, assume that $q \leq |\Sigma|^{\lambda-1}$; otherwise the claim is trivial. Consider games G_1 – G_4 in Fig. 9. Game G_1 corresponds to game \mathbf{Ideal}_H and game G_4 corresponds to game \mathbf{PRI}_H . We explain the game chain up to the terminal one. Game G_2 is identical to game G_1 , except that in procedure Enc, it ensures that ciphertexts C are distinct. Partition the encryption queries based on the nonce, the associated data, and the size of the message. Suppose that in game G_1 we have p partitions of size $s_1, \dots, s_p \geq 1$. Games G_1 and G_2 are identical-until-bad, and thus

$$\begin{aligned} |\Pr[\mathcal{A}^{G_1} \Rightarrow 1] - \Pr[\mathcal{A}^{G_2} \Rightarrow 1]| &\leq \Pr[\mathcal{A}^{G_1} \text{ sets bad}] \\ &\leq \sum_{i=1}^p \frac{s_i(s_i-1)}{|\Sigma|^{m_{\min}+\lambda+1}} \\ &= \sum_{i=1}^p \frac{(s_i-1)^2 + (s_i-1)}{|\Sigma|^{m_{\min}+\lambda+1}} \leq \frac{r^2 + r}{|\Sigma|^{m_{\min}+\lambda+1}}; \end{aligned}$$

the last inequality is due to the fact that $(s_1-1) + \dots + (s_p-1) = r$. Game G_3 is a simplified version of game G_2 ; the change is conservative. Game G_4 is identical to game G_3 , except that in procedure Dec, it samples a λ -character string v and returns a non- \perp answer if $v = \mathbf{0}^\lambda$, where $\mathbf{0}$ is a canonical point in Σ . Let L' be the multiset of $|C|$ in \mathcal{A} 's decryption queries in game G_4 , and let L be the multiset $\{\ell \mid \ell \geq 0 \text{ and } \ell + \lambda \in L'\}$. Then

$$\begin{aligned} |\Pr[\mathcal{A}^{G_3} \Rightarrow 1] - \Pr[\mathcal{A}^{G_4} \Rightarrow 1]| &\leq \Pr[\mathcal{A}^{G_3} \text{ sets bad}] \\ &\leq \sum_{\ell \in L} \frac{|\Sigma|^\ell}{|\Sigma|^{\ell+\lambda} - q} \\ &= \sum_{\ell \in L} \frac{1}{|\Sigma|^\lambda - (q/|\Sigma|^\ell)} \\ &\leq \sum_{\ell \in L} \frac{1}{|\Sigma|^\lambda - q} \leq \frac{q}{|\Sigma|^\lambda - q} \leq \frac{2q}{|\Sigma|^\lambda}; \end{aligned}$$

<pre> proc Enc(N, A, M) Games G_1 / G_2 $\ell \leftarrow M$; $C \leftarrow \Sigma^{\ell+\lambda}$ if $C \in \text{Ran}_{N,A,\ell}$ then bad \leftarrow true; $C \leftarrow \Sigma^{\ell+\lambda} \setminus \text{Ran}_{N,A,\ell}$ $\text{Ran}_{N,A,\ell} \leftarrow \text{Ran}_{N,A,\ell} \cup \{C\}$ $\text{Dom}_{N,A} \leftarrow \text{Dom}_{N,A} \cup \{(M, \mathbf{0}^\lambda)\}$ return C proc Dec(N, A, λ, C) if $C < \lambda$ then return \perp $\ell \leftarrow C - \lambda$ $(M, v) \leftarrow (\Sigma^\ell \times \Sigma^\lambda) \setminus \text{Dom}_{N,A}$ $\text{Dom}_{N,A} \leftarrow \text{Dom}_{N,A} \cup \{(M, v)\}$ return \perp </pre>	<pre> proc Enc(N, A, M) Games G_3 / G_4 $\ell \leftarrow M$; $C \leftarrow \Sigma^{\ell+\lambda}$ $\text{Dom}_{N,A} \leftarrow \text{Dom}_{N,A} \cup \{(M, \mathbf{0}^\lambda)\}$ return C proc Dec(N, A, C) if $C < \lambda$ then return \perp $\ell \leftarrow C - \lambda$ $(M, v) \leftarrow (\Sigma^\ell \times \Sigma^\lambda) \setminus \text{Dom}_{N,A}$ $\text{Dom}_{N,A} \leftarrow \text{Dom}_{N,A} \cup \{(M, v)\}$ if $v = \mathbf{0}^\lambda$ then bad \leftarrow true; return M return \perp </pre>
--	---

Fig. 9. Games used to prove Theorem 1. Here $\mathbf{0}$ is a canonical element of Σ . Games G_2 and G_4 contain the boxed statements, but games G_1 and G_3 do not.

the last inequality is due to the assumption that $q \leq |\Sigma|^{\lambda-1}$. Summing up,

$$\begin{aligned} |\Pr[\mathcal{A}^{\text{Ideal}_\Pi} \Rightarrow 1] - \Pr[\mathcal{A}^{\text{PRI}_\Pi} \Rightarrow 1]| &\leq \sum_{i=1}^3 |\Pr[\mathcal{A}^{G_i} \Rightarrow 1] - \Pr[\mathcal{A}^{G_{i+1}} \Rightarrow 1]| \\ &\leq \frac{r^2 + r}{|\Sigma|^{\lambda+m_{\min}+1}} + \frac{2q}{|\Sigma|^\lambda} \end{aligned}$$

as claimed.

B.2 Proof of Theorem 2

The reduction \mathcal{R} creates from \mathcal{A} the adversary \mathcal{B} as follows. It runs \mathcal{A} . When the latter makes an encryption query (N, A, λ, M) , if $v(M) = 1$ then the former sends the same query to its encryption oracle and returns the answer to \mathcal{A} ; otherwise it returns \perp . When \mathcal{A} makes a query (N, A, λ, C) , adversary \mathcal{B} sends the same query to its decryption oracle to get M . If $|M| = |C| - \lambda$ and $v(M) \neq 1$ then it returns $\mathbf{0} \| M$ to \mathcal{A} , where $\mathbf{0}$ is a canonical point in Σ . Otherwise, it returns M . Finally, it outputs the same guess as \mathcal{A} .

For any query (N, A, λ, C) that it receives, S' stores $(N, A, \lambda, |C|)$ in a set L_λ . It also maintains, for each (N, A, λ, ℓ) in L_λ , a set $B_{N,A,\lambda,\ell}$. Initially, $B_{N,A,\lambda,\ell} = \Sigma^{\ell-\lambda} \setminus \mathcal{M}_v$. The simulator S' works by running the simulator S . For each query (N, A, λ, C) , the simulator S' tosses a biased coin, heads landing land with probability $|B_{N,A,\lambda,\ell}|/(|B_{N,A,\lambda,\ell}| + |\Sigma|^\ell - |\Sigma|^{\ell-\lambda})$, where $\ell = |C|$. If heads shows up, simulator S' will sample $M \leftarrow B_{N,A,\lambda,\ell}$, remove M from $B_{N,A,\lambda,\ell}$, and return M . Otherwise, it runs S on query (N, A, λ, C) and output whatever S returns. Then

$$\begin{aligned} \Pr[\mathcal{A}^{\text{REAL}_{\Pi_v, S'}} \Rightarrow 1] &= \Pr[\mathcal{B}^{\text{REAL}_{\Pi, S}} \Rightarrow 1], \text{ and} \\ \Pr[\mathcal{A}^{\text{RAE}_{\Pi_v, S'}} \Rightarrow 1] &= \Pr[\mathcal{B}^{\text{RAE}_{\Pi, S}} \Rightarrow 1]. \end{aligned}$$

Subtracting, we get $\mathbf{Adv}_{\Pi, S}^{\text{rae}}(\mathcal{B}) = \mathbf{Adv}_{\Pi_v, S'}^{\text{rae}}(\mathcal{A})$.

C An Insecure Variant of AEZ-core

Numerous variants of AEZ-core were considered to arrive at AEZ-core. Most simplifications of the final version do not work. As an example, consider trying to cheapen the design by using $c_i \cdot f_{\text{aa},1}(S)$ instead of $f_{\text{aa},i}(S)$ to whiten the middle of each Feistel network, where each c_i is a public constant, and the dot is the multiplication in $\text{GF}(2^n)$. For example, one might hope this works for $c_i = 1$ or $c_i = i$. But this modification is insecure for any choice of c_i values.

For each $L \subseteq \{1, \dots, n+1\}$ let $\theta(L) = \bigoplus_{i \in L} c_i$. Let $D \neq \emptyset$ be a subset of $\{1, \dots, n+1\}$ such that $\theta(D) = 0^n$. Such a set D must exist. Assume to the contrary that $\theta(L) \neq 0^n$ for all nonempty $L \subseteq \{1, \dots, n+1\}$. Then for any distinct nonempty subsets $L, L' \subseteq \{1, \dots, n+1\}$, we have $\theta(L) \neq \theta(L')$. This means that for $2^{n+1}-1$ nonempty subsets $L \subseteq \{1, \dots, n+1\}$ we have $2^{n+1}-1 > 2^n$ corresponding distinct elements $\theta(L)$ of $\text{GF}(2^n)$, which is a contradiction.

We now describe an attack to the modified AEZ-core. Our attack only uses strings of length $\ell = 2n(n+3)$. Let M and \tilde{M} be arbitrary distinct ℓ -bit strings such that they agree everywhere except the last two blocks. Query M and \tilde{M} to the first oracle to get answers C and \tilde{C} respectively. In the real game, we'll have $X_i = \tilde{X}_i$ and $\tilde{Y}_i = Y_i \oplus (c_i \cdot (S \oplus \tilde{S}))$ for every $1 \leq i \leq n+2$. Next, let C^* be the “mixed” ciphertext such that, for every $1 \leq i \leq n+3$, the $(2i-1)$ 'th and $2i$ 'th blocks of C^* are the same as those of \tilde{C} if $i \in D$, otherwise C^* would borrow the corresponding two blocks of C . Query C^* to the second oracle to get an answer M^* . Let $\overline{D} = \{1, \dots, n+2\} \setminus D$. In the real game, the query C^* will generate $Y_i^* = \tilde{Y}_i$ for every $i \in D$, and $Y_i^* = Y_i$ for every $i \in \overline{D}$. Then

$$Y^* = \bigoplus_{i \in D} \tilde{Y}_i \oplus \bigoplus_{j \in \overline{D}} Y_j = Y \oplus \bigoplus_{i \in D} ((S \oplus \tilde{S}) \cdot c_i) = Y .$$

Consequently, $S^* = S$ and thus M^* and M agree at the $(2n+3)$ th and $(2n+4)$ th blocks. The latter event happens with probability at most 2^{-n} in the random game. Hence this attack wins with advantage at least $1 - 2^{-n}$.

References

1. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. Cryptology ePrint report 2014/144, February 25, 2014
2. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 424–443. Springer, Heidelberg (2013)
3. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: Online ciphers and the hash-CBC construction. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 292–309. Springer, Heidelberg (2001)
4. Bellare, M., Rogaway, P.: On the construction of variable-input-length ciphers. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 231–244. Springer, Heidelberg (1999)

5. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: how to exploit nonces or redundancy in plaintexts for efficient cryptography. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 317–330. Springer, Heidelberg (2000)
6. Bellare, M., Rogaway, P., Spies, T.: The FFX mode of operation for format-preserving encryption. Draft 1.1. Submission to NIST, February 20, 2010
7. Bernstein, D.: Cryptographic competitions: CAESAR call for submissions, final, January 27, 2014. <http://competitions.cr.yp.to/caesar-call.html>
8. Black, J., Cochran, M.: MAC reforgeability. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 345–362. Springer, Heidelberg (2009)
9. Black, J.A., Rogaway, P.: Ciphers with arbitrary finite domains. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 114–130. Springer, Heidelberg (2002)
10. Boldyreva, A., Degabriele, J., Paterson, K., Stam, M.: On symmetric encryption with distinguishable decryption failures. Cryptology ePrint Report 2013/433 (2013)
11. Chakraborty, D., Nandi, M.: An improved security bound for HCTR. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 289–302. Springer, Heidelberg (2008)
12. Chakraborty, D., Sarkar, P.: HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. IEEE Transactions on Information Theory **54**(4), 1683–1699 (2008)
13. Chakraborty, D., Sarkar, P.: A new mode of encryption providing a tweakable strong pseudo-random permutation. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 293–309. Springer, Heidelberg (2006)
14. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag, Heidelberg (2002)
15. Daemen, J., Rijmen, V.: A new MAC construction ALRED and a specific instance ALPHA-MAC. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 1–17. Springer, Heidelberg (2005)
16. Daemen, J., Rijmen, V.: The Pelican MAC function. Cryptology ePrint report 2005/088 (2005)
17. Dworkin, M.: Recommendation for block cipher modes of operation: methods for format-preserving encryption. NIST Special Publication 800–38G: Draft, July 2013
18. Ferguson, N.: Authentication weaknesses in GCM. Manuscript, May 20, 2005
19. Fisher, R., Yates, F.: Statistical tables for biological, agricultural and medical research. Oliver & Boyd, London (1938)
20. Fleischmann, E., Forler, C., Lucks, S.: McOE: a family of almost foolproof on-line authenticated encryption schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
21. Fouque, P., Joux, A., Martinet, G., Valette, F.: Authenticated on-line encryption. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 145–159. Springer, Heidelberg (2004)
22. Halevi, S.: EME*: extending EME to handle arbitrary-length messages with associated data. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 315–327. Springer, Heidelberg (2004)
23. Halevi, S.: Invertible universal hashing and the TET encryption mode. Cryptology ePrint report 2007/014
24. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 292–304. Springer, Heidelberg (2004)
25. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003)

26. Hoang, V.T., Krovetz, T., Rogaway, P.: AEZ v3: authenticated encryption by enciphering. CAESAR submission (2014)
27. Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer, Heidelberg (2001)
28. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption: AEZ and the problem that it solves. Cryptology ePrint report 2014/793, January 2015 (Full version of this paper)
29. IEEE. 1619.2-2010 - IEEE standard for wide-block encryption for shared storage media. IEEE press (2010)
30. Kaliski Jr., B.S., Rivest, R.L., Sherman, A.T.: Is DES a Pure Cipher? (Results of more cycling experiments on DES). In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 212–226. Springer, Heidelberg (1986)
31. Keliher, L., Sui, J.: Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. IET Information Security 1(2), 53–57 (2007)
32. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)
33. Krawczyk, H.: Cryptographic extraction and key derivation: the HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg (2010)
34. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
35. McGrew, D.A., Fluhrer, S.R.: The security of the extended codebook (XCB) mode of operation. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 311–327. Springer, Heidelberg (2007)
36. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 275–292. Springer, Heidelberg (2014)
37. Minematsu, K., Tsunoo, Y.: Provably secure MACs from differentially-uniform permutations and AES-based implementations. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 226–241. Springer, Heidelberg (2006)
38. Nandi, M.: Improving upon HCTR and matching attacks for Hash-Counter-Hash approach. Cryptology ePrint report 2008/090, February 28, 2008
39. Naor, M., Reingold, O.: On the construction of pseudo-random permutations: Luby-Rackoff revisited. Journal of Cryptology **12**(1), 29–66 (1999)
40. Naor, M., Reingold, O.: The NR mode of operation. Undated manuscript realizing the mechanism of [39]
41. Patarin, J.: Generic attacks on feistel schemes. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 222–238. Springer, Heidelberg (2001)
42. Patel, S., Ramzan, Z., Sundaram, G.S.: Efficient constructions of variable-input-length block ciphers. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 326–340. Springer, Heidelberg (2004)
43. Patarin, J.: Security of balanced and unbalanced Feistel schemes with linear non equalities. Cryptology ePrint report 2010/293, May 2010
44. Patarin, J.: Security of random feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 106–122. Springer, Heidelberg (2004)

45. Patarin, J., Gittins, B., Treger, J.: Increasing block sizes using feistel networks: the example of the AES. In: Naccache, D. (ed.) Cryptography and Security: From Theory to Applications. LNCS, vol. 6805, pp. 67–82. Springer, Heidelberg (2012)
46. Percival, C.: Stronger key derivation via sequential memory-hard functions. The BSD Conference (BSDCan), May 2009
47. Reyhanitabar, R., Vizár, D.: Careful with misuse resistance of online AEAD. Unpublished manuscript distributed on the `crypto-competitions` mailing list. August 24, 2014
48. Rogaway, P.: Authenticated-encryption with associated-data. In: ACM CCS 2002, pp. 98–107. ACM Press (2002)
49. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
50. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: A block-cipher mode of operation for efficient authenticated encryption. In: ACM CCS, pp. 196–205 (2001)
51. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
52. Sarkar, P.: Efficient tweakable enciphering schemes from (block-wise) universal hash functions. Cryptology ePrint report 2008/004
53. Sarkar, P.: Improving upon the TET mode of operation. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 180–192. Springer, Heidelberg (2007)
54. Sarkar, P.: Tweakable enciphering schemes using only the encryption function of a block cipher. Cryptology ePrint report 2009/216
55. Schroeppel, R.: Hasty Pudding Cipher Specification. AES candidate submitted to NIST, June 1998. <http://richard.schroeppel.name/hpc/hpc-spec> (revised May 1999)
56. Shrimpton, T., Terashima, R.S.: A modular framework for building variable-input-length tweakable ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 405–423. Springer, Heidelberg (2013)
57. Simplício, M., Barbuda, P., Barreto, P., Carvalho, T., Margi, C.: The MARVIN message authentication code and the LETTERSOUP authenticated encryption scheme. Security and Communications Networks **2**(2), 165–180 (2009)
58. Struik, R.: AEAD ciphers for highly constrained networks. DIAC 2013 presentation, August 13, 2013
59. Wang, P., Feng, D., Lin, C., Wu, W.: Security of truncated MACs. In: Yung, M., Liu, P., Lin, D. (eds.) Inscript 2008. LNCS, vol. 5487, pp. 96–114. Springer, Heidelberg (2009)
60. Wang, P., Feng, D., Wu, W.: HCTR: a variable-input-length enciphering mode. In: Feng, D., Lin, D., Yung, M. (eds.) CISC 2005. LNCS, vol. 3822, pp. 175–188. Springer, Heidelberg (2005)
61. Yao, F., Yin, Y.L.: Design and analysis of password-based key derivation functions. IEEE Trans. on Information Theory **51**(9), 3292–3297 (2005)

On the Behaviors of Affine Equivalent Sboxes Regarding Differential and Linear Attacks

Anne Canteaut^(✉) and Joëlle Roué

Inria, Project-team SECRET, Rocquencourt, France
`{Anne.Canteaut,Joelle.Roue}@inria.fr`

Abstract. This paper investigates the effect of affine transformations of the Sbox on the maximal expected differential probability MEDP and linear potential MELP over two rounds of a substitution-permutation network, when the diffusion layer is linear over the finite field defined by the Sbox alphabet. It is mainly motivated by the fact that the 2-round MEDP and MELP of the AES both increase when the AES Sbox is replaced by the inversion in \mathbf{F}_{2^8} . Most notably, we give new upper bounds on these two quantities which are not invariant under affine equivalence. Moreover, within a given equivalence class, these new bounds are maximal when the considered Sbox is an involution. These results point out that different Sboxes within the same affine equivalence class may lead to different two-round MEDP and MELP. In particular, we exhibit some examples where the basis chosen for defining the isomorphism between \mathbf{F}_2^m and \mathbf{F}_{2^m} affects these values. For Sboxes with some particular properties, including all Sboxes of the form $A(x^s)$ as in the AES, we also derive some lower and upper bounds for the 2-round MEDP and MELP which hold for any MDS linear layer.

Keywords: Sboxes · Affine equivalence · Differential cryptanalysis · Linear cryptanalysis · AES

1 Introduction

Cryptographic functions, including the so-called Sboxes, are usually classified up to affine equivalence (see *e.g.* [6, 11, 34]) since many of the relevant cryptographic properties are invariant under affine transformations. Indeed, both Sboxes S and $A_2 \circ S \circ A_1$, where A_1 and A_2 are two affine permutations, have the same algebraic degree, the same non-linearity (even the same square Walsh spectrum) and the same differential uniformity (even the same differential spectrum), which are the usual criteria measuring the resistance of an Sbox against higher-order differential attacks [29, 31], linear cryptanalysis [37, 44] and differential cryptanalysis [5] respectively. However, it is well-known that equivalent Sboxes may have different implementation costs and may also provide different

Partially supported by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011.

security levels. For instance, the number of terms in their polynomial representations may highly vary within an equivalent class. This has motivated the choice of the AES Sbox: it corresponds to the inversion in \mathbf{F}_{2^8} , which is a power permutation with the best known resistance against the previously mentioned attacks; but this power permutation is then composed with an \mathbf{F}_2 -affine permutation of \mathbf{F}_2^8 which makes its polynomial representation much more complex. Composing the inverse function with an affine permutation then thwarts potential attacks exploiting a simple algebraic representation of the Sbox, like an extremely sparse polynomial. Some other relevant properties (usually of minor importance) are also affected by composition with affine transformations, like the number of fixed points and the bitwise branch number [43].

But, when focusing on statistical attacks, especially on differential and linear cryptanalyses, the Sboxes within the same equivalence class are often considered to have similar behaviors. The main reason is that all known upper bounds on the maximal expected differential probability, and on the maximal expected square correlation (aka maximum expected linear potential) [41] are invariant under the affine transformations of the Sbox. However, the exact values of these two quantities for two rounds of the AES have been computed by Keliher and Sui with a sophisticated pruning algorithm [28], and it appears that the values obtained for the multiplicative inverse in \mathbf{F}_{2^8} and for the original AES Sbox are different, while these two Sboxes belong to the same equivalence class. Going further in the analysis, Daemen and Rijmen have then determined the expected probabilities of all two-round differentials with 5 or 6 active Sboxes in the AES for both Sboxes [20]. After this analysis, they have even conjectured that, for any number of rounds, the maximal expected differential probability of the AES is always higher with the inversion in \mathbf{F}_{2^8} than with the AES Sbox [17]. The aim of this paper is then to have a better understanding of this phenomenon. For instance, we would like to determine whether these different behaviors originate from the Sboxes only, independently of the choice of the diffusion layer, or not. One of our main motivations is to help the designers choose an Sbox within a given equivalence class. Indeed, in most situations, some appropriate equivalence classes are known (e.g. 4-bit permutations are classified up to affine equivalence [34]) and the search is often restricted to these classes.

Our Contribution. In this paper, we investigate the maximal expected differential probability MEDP and linear potential MELP over two rounds of an SPN. We focus on diffusion layers which are linear over the field of size 2^m , where m is the number of bits of the Sbox, exactly as in the AES and several other ciphers like LED [25], KLEIN [24], mCrypton [35], Prøst [27]... We give a new upper bound on the two-round MEDP and MELP which supersedes the best previous result [41], and which is not invariant under affine equivalence. This result is combined with the lower bounds corresponding to some minimum-weight differentials (or linear masks). We are then able to exhibit different behaviors regarding differential and linear attacks on two rounds depending on the choice of the Sbox within a given equivalence class. This includes some unexpected differences since we point out that, for a given m -bit Sbox, the choice of the basis used for

defining the finite field in the description of the linear layer may also affect the value of the two-round MEDP or MELP. This is due to the \mathbf{F}_{2^m} -linearity of the mixing layer.

More interestingly from the designers' viewpoint, for some classical families of Sboxes including all functions of the form $A(x^s)$ or $(A(x))^s$ where A is an affine function, like the AES Sbox, our results yield some lower and upper bounds on the two-round MEDP and MELP which are independent of the choice of the MDS linear layer. In other words, we show that, for these families of Sboxes, the two-round MEDP and MELP are two quantities which essentially depend on the Sbox only. Therefore, the designer can choose an Sbox and get a precise estimation of the corresponding two-round MEDP and MELP, while all previous methods [28] involved the specifications of both the Sbox and the diffusion layer together. As an illustration, we prove that the previously known upper bounds on MEDP_2 and MELP_2 due to Park et al. [41] are always tight for the multiplicative inverse over \mathbf{F}_{2^m} and for any MDS linear layer. In other words, the inversion is the mapping within its equivalence class which has the highest two-round MEDP and MELP, independently of the choice of the MDS linear layer. This situation mainly originates from the fact that this Sbox is an involution.

2 Maximum Expected Differential Probability and Linear Potential for Substitution-Permutation Networks

2.1 Substitution-Permutation Networks

One of the most widely-used constructions for iterated block ciphers is the so-called key-alternating construction [15, 18] (aka iterated Even-Mansour construction), which consists of an alternation of key-independent (usually similar) permutations and of round-key additions. The round permutation usually follows the principles introduced by Shannon. It is decomposed into a nonlinear substitution function Sub which provides confusion, and a linear permutation which provides diffusion¹. In order to reduce the implementation cost of the substitution layer, which is usually the most expensive part of the cipher in terms of circuit complexity, a usual choice for Sub consists in concatenating several copies of a permutation S which operates on a much smaller alphabet. In the whole paper, we will concentrate on such block ciphers, and use the following notation to describe the corresponding round permutation.

Definition 1. Let m and t be two positive integers. Let S be a permutation of \mathbf{F}_2^m and M be a linear permutation of \mathbf{F}_2^{mt} . Then, $\text{SPN}(m, t, S, M)$ denotes any substitution-permutation network defined over \mathbf{F}_2^{mt} whose substitution function consists of the concatenation of t copies of S and whose diffusion function corresponds to M .

¹ Here, the terminology *substitution-permutation* has to be understood in a broad sense without any restriction on the linear permutation, while in some other papers, it is limited to the class of bit permutations.

For instance, up to a linear transformation, two rounds of the AES can be seen as the concatenation of four similar *superboxes* [20]. The superbox, depicted on Fig. 1, is linearly equivalent to a two-round permutation of the form $\text{SPN}(8, 4, S, M)$ where the AES Sbox S corresponds to the composition of the inversion in \mathbf{F}_{2^8} with an affine permutation A . More precisely, $S(x) = A \circ \varphi^{-1}(\varphi(x)^{254})$ where φ is the isomorphism from \mathbf{F}_2^8 into \mathbf{F}_{2^8} defined by the basis $\{1, \alpha, \alpha^2, \dots, \alpha^7\}$ with α a root of $X^8 + X^4 + X^3 + X + 1$.

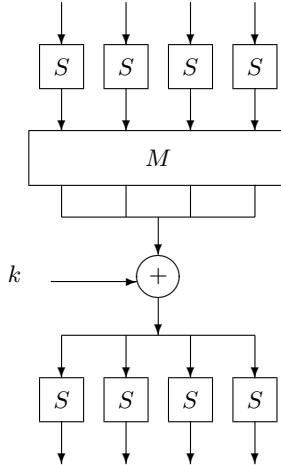


Fig. 1. The AES superbox

Differential [5] and linear [37, 44] cryptanalyses are the most prominent statistical attacks. The complexity of differential attacks depends critically on the distribution over the keys k of the probability of the differentials (a, b) , *i.e.*,

$$\text{DP}(a, b) = \Pr_X [E_k(X) + E_k(X + a) = b]$$

where E_k corresponds to the (possibly round-reduced) encryption function under key k . This probability may highly vary with the key especially when a small number of rounds is considered (see *e.g.* [32], [19, Section 8.7.2], [21], [22] and [7]). But computing the whole distribution of the probability of a differential is a very difficult task, and cryptanalysts usually focus on its expectation.

Definition 2. Let $(E_k)_{k \in \mathbf{F}_2^\kappa}$ be an r -round iterated cipher with key-size κ . Then, the expected probability of an r -round differential (a, b) is

$$\text{EDP}_r^E(a, b) = 2^{-\kappa} \sum_{k \in \mathbf{F}_2^\kappa} \Pr_X [E_k(X) + E_k(X + a) = b].$$

The maximum expected differential probability for r rounds is

$$\text{MEDP}_r^E = \max_{a \neq 0, b} \text{EDP}_r^E(a, b).$$

The index in MEDP_r^E will be omitted when the number of rounds is not specified. It is worth noticing that the MEDP is relevant for estimating the resistance against classical differential cryptanalysis, not against its variants like truncated differential attacks (which provide better attacks on the AES since the AES resists differential cryptanalysis by design).

Similarly, the resistance of a cipher against linear cryptanalysis can be evaluated by determining the distribution over the keys of the correlation of each r -round mask (u, v) :

$$C(u, v) = 2^{-n} \sum_{x \in \mathbf{F}_2^n} (-1)^{u \cdot x + v \cdot E_k(x)},$$

where n is the block-size. For a key-alternating cipher with independent round keys, the average over all keys of the correlation $C(u, v)$ is zero for any nonzero mask (u, v) (see e.g. [19, Section 7.9] or [1, Prop. 1]). Then, the major parameter investigated in this paper is the variance of the distribution of the correlation, which corresponds to the average square correlation. The way it affects the complexity of linear cryptanalysis is discussed for instance in [1, 19, 33, 38, 40].

Definition 3. Let $(E_k)_{k \in \mathbf{F}_2^\kappa}$ be an r -round iterated cipher with block-size n and key-size κ . Then, the expected square correlation (aka linear potential [40]) of an r -round mask (u, v) is

$$\text{ELP}_r^E(u, v) = 2^{-2n-\kappa} \sum_{k \in \mathbf{F}_2^\kappa} \left(\sum_{x \in \mathbf{F}_2^n} (-1)^{u \cdot x + v \cdot E_k(x)} \right)^2.$$

The maximum expected square correlation for r rounds is

$$\text{MELP}_r^E = \max_{u, v \neq 0} \text{ELP}_r^E(u, v).$$

2.2 Known Results on Two-Round MEDP and MELP

Computing the MEDP and MELP for an SPN, even for a small number of rounds, is usually non-trivial. An easier task consists in computing the expected probability of an r -round differential characteristic (*i.e.*, a collection of $(r+1)$ differences), or the expected square correlation of a linear trail (*i.e.*, a collection of $(r+1)$ linear masks). In particular, a simple upper bound on this quantity can be derived from the differential uniformity [39] (resp. the nonlinearity) of the Sbox, and from the differential (resp. linear) branch number of the linear layer. We will then extensively use the following notation for these quantities.

Definition 4. Let S be a function from \mathbf{F}_2^m into \mathbf{F}_2^m .

- For any a and b in \mathbf{F}_2^m , we define

$$\delta^S(a, b) = \#\{x \in \mathbf{F}_2^m, S(x+a) + S(x) = b\}.$$

The multi-set $\{\delta^S(a, b), a, b \in \mathbf{F}_2^m\}$ is the differential spectrum of S and its maximum $\Delta(S) = \max_{a \neq 0, b} \delta^S(a, b)$ is the differential uniformity of S .

– For any u and v in \mathbf{F}_2^m , we define

$$\mathcal{W}^S(u, v) = \sum_{x \in \mathbf{F}_2^m} (-1)^{u \cdot x + v \cdot S(x)},$$

where \cdot is the usual scalar product in \mathbf{F}_2^m . The multi-set $\{\mathcal{W}^S(u, v), u \in \mathbf{F}_2^m, v \in \mathbf{F}_2^m\}$ is the Walsh spectrum of S , and its highest magnitude $\mathcal{L}(S) = \max_{u, v \neq 0} |\mathcal{W}^S(u, v)|$ is the linearity of S .

The branch number of the diffusion layer then determines the minimum number of active Sboxes within a differential or linear trail.

Definition 5. [15] Let M be an \mathbf{F}_2 -linear permutation of $(\mathbf{F}_2^m)^t$. We associate with M the codes \mathcal{C}_M and \mathcal{C}_M^\perp of length $2t$ and size 2^t over \mathbf{F}_2^m defined by

$$\mathcal{C}_M = \{(c, M(c)), c \in (\mathbf{F}_2^m)^t\} \text{ and } \mathcal{C}_M^\perp = \{(M^*(c), c), c \in (\mathbf{F}_2^m)^t\},$$

where M^* is the adjoint of M , i.e., the linear map such that $x \cdot M(y) = M^*(x) \cdot y$ for any (x, y) . The differential branch number (resp. linear branch number) of M is the minimum distance of the code \mathcal{C}_M (resp. of \mathcal{C}_M^\perp).

From Singleton's bound, the maximum branch number of M is $(t + 1)$ and is achieved when \mathcal{C}_M is MDS. Since the codes \mathcal{C}_M and \mathcal{C}_M^\perp are dual to each other, M has optimal differential branch number if and only if it has optimal linear branch number. A simple upper bound for both the two-round MEDP and MELP can then be derived from the branch numbers of M , and from the differential uniformity and the linearity of the Sbox (see [26] and [19, Section B.2]):

$$\text{MEDP}_2 \leq (2^{-m}\Delta(S))^t \text{ and } \text{MELP}_2 \leq (2^{-m}\mathcal{L}(S))^{2t}. \quad (1)$$

This result has then been refined in [14, 41].

Theorem 1 (FSE 2003 bounds). [14, 41] Let E be a block cipher of the form $\text{SPN}(m, t, S, M)$ where M is a linear permutation with differential (resp. linear) branch number d (resp. d^\perp). Then, we have

$$\text{MEDP}_2^E \leq 2^{-md} \max \left(\max_{a \in (\mathbf{F}_2^m)^*} \sum_{\gamma \in (\mathbf{F}_2^m)^*} \delta^S(a, \gamma)^d, \max_{b \in (\mathbf{F}_2^m)^*} \sum_{\gamma \in (\mathbf{F}_2^m)^*} \delta^S(\gamma, b)^d \right),$$

$$\text{MELP}_2^E \leq 2^{-2md^\perp} \max \left(\max_{u \in (\mathbf{F}_2^m)^*} \sum_{\gamma \in (\mathbf{F}_2^m)^*} \mathcal{W}^S(u, \gamma)^{2d^\perp}, \max_{v \in (\mathbf{F}_2^m)^*} \sum_{\gamma \in (\mathbf{F}_2^m)^*} \mathcal{W}^S(\gamma, v)^{2d^\perp} \right)$$

It is worth noticing that the FSE 2003 bounds always supersede (1).

The main question is now to determine the gap between the FSE 2003 bounds and the exact values of MEDP_2 and MELP_2 for a given cipher. An interesting property is that the FSE 2003 bound is invariant under affine equivalence, i.e., under left or right composition of the Sbox with an affine permutation. Actually, the following well-known property holds.

Lemma 1. Let S be permutation of \mathbf{F}_2^m and A_1 and A_2 be two affine permutations of \mathbf{F}_2^m . Then $S' = A_2 \circ S \circ A_1$ satisfies

$$\delta^{S'}(a, b) = \delta^S(L_1(a), L_2^{-1}(b)) \text{ and } \mathcal{W}^{S'}(u, v)^2 = \mathcal{W}^S((L_1^{-1})^*(u), L_2^*(v))^2$$

where L_1 and L_2 correspond to the linear parts of A_1 and A_2 , and L^* denotes the adjoint of L .

However, while the previous bounds are invariant under affine equivalence, it appears that the exact values of MEDP₂ and MELP₂ may vary when the Sbox is composed with an affine permutation. For instance, for the AES with its original Sbox, the exact values of the two-round MEDP₂ and MELP₂ have been computed by a pruning search algorithm [28]: MEDP₂ = 53×2^{-34} and MELP₂ = $109,953,193 \times 2^{-54} \approx 1.638 \times 2^{-28}$. But, if the AES Sbox is replaced by the so-called *naive Sbox* [17], obtained by removing the affine permutation from the AES Sbox, MEDP₂ = 79×2^{-34} [20] which corresponds to the FSE 2003 bound. To our best knowledge, the exact value of MELP₂ for the naive Sbox has not been computed, but we will deduce from our results in Section 4.3 that, for the multiplicative inverse over \mathbf{F}_{2^m} and any MDS \mathbf{F}_{2^m} -linear layer, the FSE 2003 bound is always tight. In particular, for $m = 8$, MELP₂ = $192,773,764 \times 2^{-54} \approx 2.873 \times 2^{-28}$. Then, the AES Sbox provides a better resistance against differential and linear cryptanalyses for two rounds of the AES than the naive Sbox. More generally, it has been conjectured in [17, Conjecture 1] that, for any number of rounds r , MEDP _{r} is smaller for the AES Sbox than for the naive Sbox.

2.3 SPNs over \mathbf{F}_{2^m}

A special case of affine equivalent Sboxes corresponds to the mappings over \mathbf{F}_2^m which are derived from the same function over the finite field \mathbf{F}_{2^m} , but from different correspondences between \mathbf{F}_{2^m} and the vector space \mathbf{F}_2^m . Such equivalent Sboxes appear in several situations. Indeed, a simple construction for an optimal linear layer consists in choosing for M a permutation of \mathbf{F}_2^{mt} associated with a code \mathcal{C}_M which is linear over the field \mathbf{F}_{2^m} , where m is the size of the Sbox. Then, this diffusion layer has to be defined over $\mathbf{F}_{2^m}^t$, instead of \mathbf{F}_2^{mt} . To this end, we need to identify the vector space \mathbf{F}_2^m with the finite field \mathbf{F}_{2^m} by the means of an isomorphism φ associated to a basis $(\alpha_0, \dots, \alpha_{m-1})$, namely:

$$\begin{aligned} \varphi : \quad \mathbf{F}_2^m &\rightarrow \mathbf{F}_{2^m} \\ (x_0, \dots, x_{m-1}) &\mapsto \sum_{i=0}^{m-1} x_i \alpha_i . \end{aligned}$$

Then, both the Sbox and the diffusion layer can be represented as functions over the field \mathbf{F}_{2^m} by

$$\mathcal{S} = \varphi \circ S \circ \varphi^{-1} \text{ and } \mathcal{M} = \tilde{\varphi} \circ M \circ \tilde{\varphi}^{-1} ,$$

where $\tilde{\varphi}$ is the concatenation of t copies of φ . In this case, as noticed in [19, Section A.5], any r rounds of SPN(m, t, S, M) can be written as $\tilde{\varphi}^{-1} \circ \text{Add}_{k_r} \circ \dots \circ$

$\mathcal{R} \circ \text{Add}_{k_1} \circ \mathcal{R} \circ \text{Add}_{k_0} \circ \tilde{\varphi}$ where the round function $\mathcal{R} = \mathcal{M} \circ (\mathcal{S}, \dots, \mathcal{S})$ is a permutation of $(\mathbf{F}_{2^m})^t$ and Add_x denotes the addition of x in $(\mathbf{F}_{2^m})^t$. Obviously, composing by $\tilde{\varphi}$ at the beginning and by $\tilde{\varphi}^{-1}$ at the end changes neither the MEDP nor the MELP. This implies that MEDP_r^E and MELP_r^E depend on \mathcal{M} and \mathcal{S} only, i.e., on the representations of the Sbox and of the diffusion layer over \mathbf{F}_{2^m} . In particular, *the choice of the basis $(\alpha_0, \dots, \alpha_{m-1})$ has no influence on the differential and linear properties of the cipher*. For this reason, we use the following alternative notation for defining an SPN from these representations.

Definition 6. Let m and t be two positive integers. Let \mathcal{S} be a permutation of \mathbf{F}_{2^m} and \mathcal{M} be a permutation of $(\mathbf{F}_{2^m})^t$ which is linear over \mathbf{F}_{2^m} . Then, we denote by $\text{SPN}_F(m, t, \mathcal{S}, \mathcal{M})$ a substitution-permutation network defined over $(\mathbf{F}_{2^m})^t$ whose substitution function consists of the concatenation of t copies of \mathcal{S} and whose diffusion function corresponds to \mathcal{M} .

For the sake of clarity, all quantities related to the representation in the field \mathbf{F}_{2^m} will be indexed by F , and all functions defined over \mathbf{F}_{2^m} will be denoted by calligraphic letters. As pointed out in [23], the differential and linear properties of any $\text{SPN}(m, t, \mathcal{S}, \mathcal{M})$ can be equivalently studied by considering the alternative representation $\text{SPN}_F(m, t, \mathcal{S}, \mathcal{M})$. This alternative analysis then involves the differential spectrum and the Walsh spectrum of the Sbox \mathcal{S} over \mathbf{F}_{2^m} , which are related to the spectra of the corresponding function S over \mathbf{F}_2^m as follows.

Proposition 1. (see e.g. [23]) Let $(\alpha_0, \dots, \alpha_{m-1})$ be a basis of \mathbf{F}_{2^m} , and φ the corresponding isomorphism from \mathbf{F}_2^m into \mathbf{F}_{2^m} . Let S be a mapping over \mathbf{F}_2^m , and $\mathcal{S} = \varphi \circ S \circ \varphi^{-1}$. Then, for any $(\alpha, \beta) \in \mathbf{F}_{2^m}$,

$$\delta_F^{\mathcal{S}}(\alpha, \beta) = \#\{x \in \mathbf{F}_{2^m}, \mathcal{S}(x + \alpha) + \mathcal{S}(x) = \beta\} = \delta^S(\varphi^{-1}(\alpha), \varphi^{-1}(\beta))$$

$$\mathcal{W}_F^{\mathcal{S}}(\alpha, \beta) = \sum_{x \in \mathbf{F}_{2^m}} (-1)^{\text{Tr}(\alpha x + \beta \mathcal{S}(x))} = \mathcal{W}^S(\psi^{-1}(\alpha), \psi^{-1}(\beta))$$

where ψ is the isomorphism from \mathbf{F}_2^m into \mathbf{F}_{2^m} defined by the dual basis, i.e., the basis $(\beta_0, \dots, \beta_{m-1})$ such that $\text{Tr}(\alpha_i \beta_j) = 0$ if $i \neq j$ and $\text{Tr}(\alpha_i \beta_i) = 1$.

3 New Upper Bounds on the 2-Round MEDP and MELP

Now, we study the exact values of the two-round MEDP and MELP for any cipher of the form $\text{SPN}(m, t, \mathcal{S}, \mathcal{M})$ where the diffusion layer M is linear over \mathbf{F}_{2^m} , like in the AES. We aim at obtaining a better approximation of the MEDP_2 and MELP_2 by finding some improved lower and upper bounds. In particular, we would like to be able to differentiate affine equivalent Sboxes.

3.1 The New Upper Bounds

From now on, when considering a cipher of the form $\text{SPN}_F(m, t, \mathcal{S}, \mathcal{M})$, $\delta_F(\alpha, \beta)$ and $\mathcal{W}_F(\alpha, \beta)$ will denote the differential and Walsh spectra of the Sbox \mathcal{S} . The considered Sbox will be mentioned in the notation in case of ambiguity only.

Theorem 2. Let E be a block cipher of the form $\text{SPN}_F(m, t, \mathcal{S}, \mathcal{M})$ where \mathcal{M} is linear over \mathbf{F}_{2^m} and has differential (resp. linear) branch number d (resp. d^\perp). For $\mu \in \mathbf{F}_{2^m}$ and $u > 0$, we define

$$\mathcal{B}_u(\mu) = \max_{\alpha, \beta, \lambda \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \delta_F(\alpha, \gamma)^u \delta_F(\gamma\lambda + \mu, \beta)^{(d-u)}, \quad (2)$$

$$\mathcal{B}_u^\perp(\mu) = \max_{\alpha, \beta, \lambda \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \mathcal{W}_F(\alpha, \gamma)^{2u} \mathcal{W}_F(\gamma\lambda + \mu, \beta)^{2(d^\perp-u)}, \quad (3)$$

$$\mathcal{B}(\mu) = \max_{1 \leq u < d} \mathcal{B}_u(\mu) \text{ and } \mathcal{B}^\perp(\mu) = \max_{1 \leq u < d^\perp} \mathcal{B}_u^\perp(\mu).$$

Then,

$$\text{MEDP}_2^E \leq 2^{-md} \max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}(\mu) \text{ and } \text{MELP}_2^E \leq 2^{-2md^\perp} \max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}^\perp(\mu).$$

The proof is given in Appendix A. It mainly exploits the special form of the codewords in an \mathbf{F}_{2^m} -linear code (see Lemma 2 in Appendix A). In the whole paper, the proofs in the context of differential attacks and of linear attacks are similar. Actually, all results can be written in a more generic way, by replacing the $2^m \times 2^m$ matrix with coefficients $2^{-m}\delta_F(\alpha, \beta)$, $\alpha, \beta \in \mathbf{F}_{2^m}$, or with coefficients $2^{-2m}\mathcal{W}_F(\alpha, \beta)^2$, by any matrix $(\Lambda(\alpha, \beta))_{\alpha, \beta \in \mathbf{F}_{2^m}}$, such that the coefficients $\Lambda(\alpha, \beta)$ lie between 0 and 1 and satisfy $\Lambda(\alpha, 0) = \Lambda(0, \alpha) = 0$ for any nonzero α and $\sum_{\beta \in \mathbf{F}_{2^m}} \Lambda(\alpha, \beta) = \sum_{\beta \in \mathbf{F}_{2^m}} \Lambda(\beta, \alpha) = 1$ for all $\alpha \in \mathbf{F}_{2^m}$. Clearly, both normalized differential and Walsh spectra satisfy these conditions.

Computing this new bound is obviously more expensive than computing the FSE 2003 bound since we have to take the maximum of a similar quantity over all λ and μ . We will see in Section 4 that this bound simplifies in some cases, for instance for all Sboxes corresponding to the composition of a power permutation with an affine mapping, like the AES Sbox. Also, we will show that this refined bound may enable us to deduce the exact values of the MEDP_2 and MELP_2 in a much more efficient way than the ad hoc search algorithm presented in [28].

In the case of the AES Sbox over \mathbf{F}_{2^8} and $d = d^\perp = 5$, these new bounds lead to $\text{MEDP}_2 \leq 55.5 \times 2^{-34}$ instead of $\text{MEDP}_2 \leq 79 \times 2^{-34}$ for the FSE 2003 bound, and $\text{MELP}_2 \leq 31,231,767 \times 2^{-52}$ instead of $\text{MELP}_2 \leq 48,193,441 \times 2^{-52}$. This seems to be a minor improvement since there is only a factor $\rho \simeq 0.7$ between the two bounds. However, in AES-like constructions, the 2-round MEDP and MELP correspond to the average differential uniformity and linearity of the average superbox. Upper-bounds on the 4-round MEDP and MELP can then be derived from these values using (1). Then, we get a factor ρ^{d-1} (resp. $\rho^{d^\perp-1}$) between the bounds on MEDP_4 and MELP_4 .

While the FSE 2003 bound corresponds to the highest d -th power moment of a row or a column in the difference table of the Sbox (or in the square correlation table), our new bound involves together a row and a column in the table. In other words, this new bound depends on the link between some quantity (e.g. a derivative or the squared Walsh transform of a component) for \mathcal{S} , and the same

quantity for the inverse permutation \mathcal{S}^{-1} . This clearly appears when \mathcal{S} has a two-valued differential spectrum, since the expression of $\mathcal{B}(\mu)$ simplifies to

$$\mathcal{B}(\mu) = \Delta(\mathcal{S})^d \max_{\alpha, \beta, \lambda \in \mathbf{F}_{2^m}^*} \# (\text{Im}(D_\alpha \mathcal{S}) \cap [\lambda \text{Im}(D_\beta \mathcal{S}^{-1}) + \mu]) ,$$

where $D_\alpha \mathcal{S}$ denotes the derivative of \mathcal{S} at point α , *i.e.*, the function $x \mapsto \mathcal{S}(x + \alpha) + \mathcal{S}(x)$. Similarly, if \mathcal{S} is a plateaued function [45], *i.e.*, a function whose Walsh spectrum contains the values 0 and $\pm \mathcal{L}(\mathcal{S})$ only, we get

$$\mathcal{B}^\perp(\mu) = \mathcal{L}(\mathcal{S})^{2d^\perp} \max_{\alpha, \beta, \lambda \in \mathbf{F}_{2^m}^*} \# (\text{Supp}(\widehat{\mathcal{S}_\alpha^{-1}}) \cap [\lambda \text{Supp}(\widehat{\mathcal{S}_\beta}) + \mu]) ,$$

where \mathcal{S}_α denotes the Boolean function $x \mapsto \text{Tr}(\alpha \mathcal{S}(x))$, and \widehat{f} denotes the Walsh transform of f , *i.e.*, $\alpha \mapsto \sum_{x \in \mathbf{F}_{2^m}} (-1)^{\text{Tr}(f(x) + \alpha x)}$. It appears from these formulas that the cardinality of the intersection of such sets cannot exceed the cardinality of each set (equal to $2^m/\Delta(\mathcal{S})$ and $2^{2m}/\mathcal{L}(\mathcal{S})^2$ respectively), and that this maximum is obviously tight when \mathcal{S} is an involution, *i.e.*, $\mathcal{S}^{-1} = \mathcal{S}$. But when the Sbox is composed with a randomly chosen affine permutation, the two sets can be considered as independent. Then, the expected cardinality of their intersection is about $2^m \pi_\Delta^2 = 2^m/\Delta(\mathcal{S})^2$ (resp. $2^m \pi_\mathcal{L}^2 = 2^{3m}/\mathcal{L}(\mathcal{S})^4$) where $\pi_\Delta = 1/\Delta(\mathcal{S})$ is the proportion of nonzero elements within a row or a column of the difference table, and $\pi_\mathcal{L} = 2^m/\mathcal{L}(\mathcal{S})^2$ is the proportion of nonzero elements within a row or a column of the square correlation table. For instance, for an almost bent Sbox, *i.e.*, with m odd, $\Delta(\mathcal{S}) = 2$ and $\mathcal{L}(\mathcal{S}) = 2^{(m+1)/2}$, the expected cardinality of the two sets involved in the previous formulas is 2^{m-2} , while it is equal to 2^{m-1} when \mathcal{S} is an involution. More generally, our new upper bound is always smaller than or equal to the corresponding FSE 2003 bound, with equality when \mathcal{S} is an involution, as stated in the following proposition (see the proof in Appendix A).

Proposition 2. *Let \mathcal{S} be a permutation of \mathbf{F}_{2^m} and d be some positive integer. Then, each of the two upper bounds defined in Theorem 2 is less than or equal to the corresponding FSE 2003 bound. Moreover, equality holds if \mathcal{S} is an involution, since for any integer $u < d$,*

$$\max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}_u(\mu) = \mathcal{B}_u(0) = \max_{\alpha \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \delta_F(a, \gamma)^d = \max_{b \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \delta_F(\gamma, b)^d$$

$$\text{and } \max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}_u^\perp(\mu) = \mathcal{B}_u^\perp(0) = \max_{\alpha \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \mathcal{W}_F(a, \gamma)^{2d} = \max_{b \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \mathcal{W}_F(\gamma, b)^{2d}.$$

3.2 Some Lower Bounds

An interesting question is to determine whether these new bounds are optimal, in the sense that, for a given Sbox, there exists a linear layer such that the bounds are tight. Here, we exhibit some functions \mathcal{M} with optimal branch number such

that the EDP (resp. ELP) of some minimum-weight differential (resp. linear mask) over two rounds is related to the upper bound of Theorem 2. This lower bound results from the fact that the minimum-weight codewords with a given support in an \mathbf{F}_{2^m} -linear MDS code form a set of the form $\{\lambda c, \lambda \in \mathbf{F}_{2^m}^*\}$ for some fixed codeword c . Such a set is named *a bundle* in [20]. It is not difficult to observe that $\mathcal{B}(0)$ (resp. $\mathcal{B}^\perp(0)$) corresponds to the maximum EDP (resp. to the maximum ELP) of some particular minimum-weight bundles. Moreover, for any such particular bundle, a function \mathcal{M} such that $\mathcal{C}_\mathcal{M}$ contains this bundle can be constructed from some Generalized Reed-Solomon code.

Proposition 3. *Let \mathcal{S} be a permutation of \mathbf{F}_{2^m} and t be any integer such that $t \leq 2^{m-1}$. Then, there exist two \mathbf{F}_{2^m} -linear diffusion layers \mathcal{M}_1 and \mathcal{M}_2 over $\mathbf{F}_{2^m}^t$ with maximal branch number $d = t + 1$ such that any block cipher E_1 of the form $\text{SPN}_F(m, t, \mathcal{S}, \mathcal{M}_1)$ and E_2 of the form $\text{SPN}_F(m, t, \mathcal{S}, \mathcal{M}_2)$ satisfy*

$$\text{MEDP}_2^{E_1} \geq 2^{-m(t+1)} \mathcal{B}(0) \text{ and } \text{MELP}_2^{E_2} \geq 2^{-2m(t+1)} \mathcal{B}^\perp(0)$$

where $\mathcal{B}(0)$ and $\mathcal{B}^\perp(0)$ are defined as in Theorem 2.

Proof. We give here the proof for MEDP_2 only, but it is similar for MELP_2 . Actually, the proposition can be formulated in a generic way as the results in Appendix A. Let $\widehat{\alpha}, \widehat{\beta}, \widehat{\lambda} \in \mathbf{F}_{2^m}^*$ and $1 \leq \widehat{u} \leq t$ be some values such that $\sum_{\gamma \in \mathbf{F}_{2^m}^*} \delta_F(\widehat{\alpha}, \gamma)^{\widehat{u}} \delta_F(\gamma \widehat{\lambda}, \widehat{\beta})^{t+1-\widehat{u}} = \mathcal{B}(0)$. Let $a \in \mathbf{F}_{2^m}^t$ be the input difference whose first \widehat{u} coordinates equal $\widehat{\alpha}$ and whose last $(t - \widehat{u})$ coordinates equal 0. Similarly, $b \in \mathbf{F}_{2^m}^t$ denotes the output difference whose first $(t + 1 - \widehat{u})$ coordinates equal $\widehat{\beta}$ and whose last $(\widehat{u} - 1)$ coordinates equal 0. Since

$$\text{EDP}_2(a, \mathcal{M}(b)) = \sum_{c \in \mathcal{C}_\mathcal{M}} \left(\prod_{i=1}^t \delta_F(a_i, c_i) \right) \left(\prod_{j=1}^t \delta_F(c_{t+j}, b_j) \right),$$

it is equal to $\mathcal{B}(0)$ if the words of the form

$$\gamma \underbrace{(1, \dots, 1)}_{\widehat{u}}, \underbrace{0, \dots, 0}_{t-\widehat{u}}, \underbrace{\widehat{\lambda}, \dots, \widehat{\lambda}}_{t+1-\widehat{u}}, \underbrace{0, \dots, 0}_{\widehat{u}-1} \quad (4)$$

are the codewords in $\mathcal{C}_\mathcal{M}$ having the same support as (a, b) . Therefore, we aim at finding a linear MDS diffusion layer \mathcal{M} such that $\mathcal{C}_\mathcal{M}$ contains these codewords. Since $t \leq 2^{m-1}$, we can choose $2t$ distinct elements x_1, \dots, x_{2t} in \mathbf{F}_{2^m} . For any choice of $2t$ elements v_1, \dots, v_{2t} , we define the $t \times t$ matrix

$$R = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 v_1 & x_2 v_2 & \dots & x_t v_t \\ x_1^2 v_1 & x_2^2 v_2 & \dots & x_t^2 v_t \\ \vdots & & & \\ x_1^{t-1} v_1 & x_2^{t-1} v_2 & \dots & x_t^{t-1} v_t \end{bmatrix}^{-1} \times \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{t+1} v_{t+1} & x_{t+2} v_{t+2} & \dots & x_{2t} v_{2t} \\ x_{t+1}^2 v_{t+1} & x_{t+2}^2 v_{t+2} & \dots & x_{2t}^2 v_{2t} \\ \vdots & & & \\ x_{t+1}^{t-1} v_{t+1} & x_{t+2}^{t-1} v_{t+2} & \dots & x_{2t}^{t-1} v_{2t} \end{bmatrix}.$$

Then, the code $\mathcal{C}_\mathcal{M} = \{(x, xR), x \in \mathbf{F}_{2^m}^t\}$ is the *generalized Reed-Solomon code* $\text{GRS}_t(x_1, \dots, x_{2t}; v)$. It is well-known [36, Page 303] that this code is MDS and is

composed of all words of the form $(v_1 F(x_1), \dots, v_{2t} F(x_{2t}))$ where F ranges over all polynomials in $\mathbf{F}_{2^m}[X]$ of degree strictly less than t . Then, the codewords in \mathcal{C}_M having the same support as (a, b) correspond to the polynomials of degree at most $(t - 1)$ which vanish at all $(t - 1)$ points x_i for $i \notin \text{Supp}((a, b))$. Then, these polynomials can be written as $\gamma \widehat{F}(x)$, $\gamma \in \mathbf{F}_{2^m}^*$, and $\widehat{F}(x_i) \neq 0$ for $i \in \text{Supp}((a, b))$ since \widehat{F} cannot have more than $(t - 1)$ roots. Therefore, we can choose for v a vector such that $v_i = 1/\widehat{F}(x_i)$ for $1 \leq i \leq \bar{u}$ and $v_i = \widehat{\lambda}/\widehat{F}(x_i)$ for $t + 1 \leq i \leq 2t + 1 - \bar{u}$. This guarantees that the words in \mathcal{C}_M having the same support as (a, b) are the words of the form (4). It follows that

$$\begin{aligned} \text{EDP}_2(a, M(b)) &= \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\prod_{i=1}^{\bar{u}} \delta_F(\widehat{\alpha}, \gamma v_i \widehat{F}(x_i)) \right) \left(\prod_{j=1}^{t+1-\bar{u}} \delta_F(\gamma v_{t+j} \widehat{F}(x_{t+j}), \widehat{\beta}) \right) \\ &= \sum_{\gamma \in \mathbf{F}_{2^m}^*} \delta_F(\widehat{\alpha}, \gamma)^{\bar{u}} \delta_F(\gamma \widehat{\lambda}, \widehat{\beta})^{t+1-\bar{u}}. \end{aligned}$$

□

Remark 1. In some particular cases, we can find a generalized Reed-Solomon code corresponding to a linear layer M for which the two bounds hold together. Indeed, we want to construct a code \mathcal{C}_M which contains the words (4) and whose dual \mathcal{C}_M^\perp contains the words

$$\gamma(\underbrace{0, \dots, 0}_{t-\bar{u}}, \underbrace{\bar{\lambda}, \dots, \bar{\lambda}}_{\bar{u}}, \underbrace{0, \dots, 0}_{\bar{u}-1}, \underbrace{1, \dots, 1}_{t+1-\bar{u}})$$

for some given $\bar{\lambda}$ and \bar{u} . But the dual of $\text{GRS}_t(x_1, \dots, x_{2t}; v)$ is another generalized Reed-Solomon code, $\text{GRS}_t(x_1, \dots, x_{2t}; w)$ with $w_i^{-1} = v_i \prod_{j \neq i} (x_i + x_j)$. In particular, if $\bar{u} + \bar{u} = t$, we can find a vector (v_1, \dots, v_{2t}) such that both conditions hold together. This situation occurs for instance when S is an involution since $B(0)$ (resp. $B^\perp(0)$) is attained for all $\bar{u} < d$ (resp. for all $\bar{u} < d^\perp$).

An interesting situation is the case where the maximum over all $\mu \in \mathbf{F}_{2^m}$ of $B(\mu)$ (resp. of $B^\perp(\mu)$) is attained for $\mu = 0$. Then there exists some M for which the upper bound from Theorem 2 is tight for $\text{SPN}_F(m, t, S, M)$, implying that it is impossible to find a general better bound which depends on S and t only. This situation occurs in particular for any involutational Sbox. Indeed, by combining Prop. 2 and 3, we deduce that, for any involutational Sbox and any $t \leq 2^{m-1}$, there exists a linear layer over $\mathbf{F}_{2^m}^t$ such that the exact values of MEDP_2 and of MELP_2 are equal to the FSE 2003 bounds.

Corollary 1. *Let S be an involution of \mathbf{F}_{2^m} and t be any integer with $t \leq 2^{m-1}$. Then, there exist an \mathbf{F}_{2^m} -linear diffusion layer M over $\mathbf{F}_{2^m}^t$ with maximal branch number such that $\text{SPN}_F(m, t, S, M)$ satisfies*

$$\text{MEDP}_2 = \max_{a \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\frac{\delta_F(a, \gamma)}{2^m} \right)^{(t+1)} = \max_{b \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\frac{\delta_F(\gamma, b)}{2^m} \right)^{(t+1)}$$

$$\text{and } \text{MELP}_2 = \max_{a \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\frac{\mathcal{W}_F(a, \gamma)}{2^m} \right)^{2(t+1)} = \max_{b \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\frac{\mathcal{W}_F(\gamma, b)}{2^m} \right)^{2(t+1)}.$$

Proof. We know from Prop. 2 that for any u , $\max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}_u(\mu) = \mathcal{B}_u(0) = \mathcal{B}(0)$, and $\max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}_u(\mu)^\perp = \mathcal{B}_u(0)^\perp = \mathcal{B}(0)$. Moreover, all these values are equal to the FSE 2003 bounds. By combining Th. 2 and Prop. 3, we deduce the existence of some linear layers for which MEDP₂ (resp. MELP₂) are lower- and upper-bounded by $\mathcal{B}(0)$ (resp. $\mathcal{B}^\perp(0)$). Moreover, we have proved in Prop. 2 that $\mathcal{B}(0)$ (resp. $\mathcal{B}^\perp(0)$) is attained for all values of u . This is a case where we can construct a GRS code satisfying the conditions for MEDP₂ and MELP₂ together. \square

Example 1. The Prøst permutation over \mathbf{F}_2^{16d} , $d \geq 1$, is the core function of several AEAD-schemes submitted to the CAESAR competition [27]. This permutation is of the form SPN(4, 4d, S, M) where S is a 4-bit involution named **SubRows** and M corresponds to the composition of two linear permutations, **MixSlices** and **ShiftPlanes**. The round-constant addition is omitted here since it does not have any impact in our context. Similarly to the AES, two consecutive rounds of the Prøst permutation can be seen as the parallel application of d copies of a superbox defined over \mathbf{F}_{16} . This superbox corresponds to two **SubRows** layers separated by a **MixSlices** transformation. Moreover, even if it is not mentioned in the design rationale, it can be checked that **MixSlices** is linear over \mathbf{F}_{16} if \mathbf{F}_{16} is identified with \mathbf{F}_2^4 by the following isomorphism:

$$\varphi : (x_0, \dots, x_3) \mapsto x_1 + \alpha x_2 + \alpha^2 x_3 + \alpha^3 x_0$$

where α is a root of $X^4 + X^3 + 1$. Indeed, the function defined over \mathbf{F}_{16}^4 by $\mathcal{M} = \tilde{\varphi} \circ \text{MixSlices} \circ \tilde{\varphi}^{-1}$ corresponds to the multiplication by

$$\begin{pmatrix} 1 & \alpha & \alpha + \alpha^2 & \alpha^2 \\ \alpha & 1 & \alpha^2 & \alpha + \alpha^2 \\ \alpha + \alpha^2 & \alpha^2 & 1 & \alpha \\ \alpha^2 & \alpha + \alpha^2 & \alpha & 1 \end{pmatrix}.$$

The previous framework then directly applies². In particular, since the Sbox is an involution, our bounds are equal to the FSE 2003 bounds (Prop. 2): the Prøst permutation with any \mathbf{F}_2 -linear MDS **MixSlices** transformation satisfies

$$\text{MEDP}_2 \leq 2^{-8} \text{ and } \text{MELP}_2 \leq 2^{-8}.$$

These bounds are tight as stated in Corollary 1: using the construction described in the proof of Prop. 3, we obtain that the following matrix over \mathbf{F}_{16} (with the previously described representation) leads to a variant of the Prøst permutation with $\text{MEDP}_2 = \text{MELP}_2 = 2^{-8}$:

² We here focus on the MEDP and MELP of the SPN with the same building blocks as the Prøst permutation, but these expectations do not provide any direct information on the security of the Prøst permutation in which the key is fixed.

$$\begin{pmatrix} \alpha^2 + \alpha + 1 & \alpha^3 + \alpha & \alpha^3 + \alpha + 1 & 1 \\ \alpha + 1 & \alpha^3 + \alpha^2 + \alpha & \alpha^2 + \alpha + 1 & 1 \\ \alpha^2 + 1 & \alpha^3 + \alpha^2 + 1 & \alpha^3 & 1 \\ \alpha^2 & \alpha^3 + \alpha^2 & \alpha^3 + 1 & 1 \end{pmatrix}.$$

This implies that, for this particular Sbox, the **MixSlices** transformation must be chosen with care to guarantee small MEDP₂ and MELP₂. Instead, for some Sboxes within the same equivalence class as **SubRows**, we can guarantee that, for any \mathbf{F}_{16} -linear MDS **MixSlices**, $\text{MEDP}_2 \leq 3 \times 2^{-10}$. This does not make a big difference in the case of Prøst since the alphabet is small and the exact MEDP₂ and MELP₂ can be easily computed. For instance, for the **MixSlices** transformation chosen by the designers, we have $\text{MEDP}_2 = 3 \times 2^{-11}$ and $\text{MELP}_2 = 81 \times 2^{-16}$. However, for Sboxes over \mathbf{F}_{2^8} , computing the exact MEDP₂ and MELP₂ is rather expensive and obtaining a better upper bound is very helpful.

3.3 Influence of the Field Representation

Clearly, there is no reason why the two-round MEDP or MELP should be the same for affine equivalent Sboxes in general. Then, it makes sense that our new bounds are not invariant under affine equivalence. More surprisingly, by combining the upper bound from Theorem 2 with the lower bound provided by Prop. 3, we can exhibit some examples showing that the choice of the field \mathbf{F}_{2^m} , *i.e.*, the choice of the isomorphism φ between \mathbf{F}_2^m and \mathbf{F}_{2^m} , may influence the value of the MEDP and MELP.

Example 2. Let us consider 2 rounds of a cipher of the form SPN(4, 4, S, M), where S is one of the permutations of \mathbf{F}_2^4 used in the PRINCE-family [9], namely permutation S_6 in [10, Table 3]:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	0	1	2	13	4	7	15	6	8	14	11	10	9	3	12	5

where each element in \mathbf{F}_2^4 is here represented as an integer between 0 and 15. This Sbox is differentially 4-uniform and has linearity $\mathcal{L}(S) = 8$. For this Sbox, the FSE 2003 bound gives $\text{MEDP}_2^E \leq 34 \times 2^{-14}$, for any \mathbf{F}_2 -linear diffusion layer M over \mathbf{F}_2^{16} with branch number 5. If we now consider a diffusion layer \mathcal{M} with branch number 5 which is linear over \mathbf{F}_{2^4} where \mathbf{F}_{2^4} is identified with \mathbf{F}_2^4 by the basis $\{1, \alpha, \alpha^2, \alpha^3\}$ where α is a root of the irreducible polynomial $X^4 + X^3 + X^2 + X + 1$, we get from Theorem 2 that

$$\text{MEDP}_2^E \leq 33 \times 2^{-14},$$

and this inequality holds for any such function \mathcal{M} . However, we can now consider a permutation \mathcal{M}' which is linear over \mathbf{F}_{2^4} , but where the isomorphism between \mathbf{F}_2^4 and \mathbf{F}_{2^4} is defined by a different basis, namely $\{1, \beta, \beta^2, \beta^3\}$ where β is a root of the primitive polynomial $X^4 + X + 1$. Then, the value $\mathcal{B}(0)$ involved in

Prop. 3 equals 17×2^7 , implying that there exists an \mathbf{F}_{2^4} -linear function \mathcal{M}' with branch number 5 such that

$$\text{MEDP}_2^E = 34 \times 2^{-14},$$

which is strictly higher than the upper bound we have for any MDS diffusion layer \mathbf{F}_{2^4} -linear where \mathbf{F}_{2^4} is defined with the basis $\{1, \alpha, \alpha^2, \alpha^3\}$.

There is no contradiction here since the two theorems apply to the representations of the Sbox and of the diffusion function over \mathbf{F}_{2^4} only. Here, we have proved that there is a particular \mathcal{M}' such that $\text{SPN}(4, 4, S, \tilde{\psi}^{-1} \circ \mathcal{M}' \circ \tilde{\psi})$ has $\text{MEDP}_2 = 34 \times 2^{-14}$, where $\tilde{\psi}$ is the concatenation of 4 copies of the isomorphism ψ from \mathbf{F}_2^4 into \mathbf{F}_{2^4} defined by the basis $\{1, \beta, \beta^2, \beta^3\}$. But if we consider the basis $\{1, \alpha, \alpha^2, \alpha^3\}$ and the corresponding isomorphism φ , Th. 2 provides a bound for all $\text{SPN}_F(4, 4, \varphi \circ S \circ \varphi^{-1}, \mathcal{M})$ which does not include the previous case because the permutation defined by $\mathcal{M} = \tilde{\psi} \circ \tilde{\psi}^{-1} \circ \mathcal{M}' \circ \tilde{\psi} \circ \tilde{\psi}^{-1}$ is not linear over \mathbf{F}_{2^4} , since $(\psi \circ \varphi^{-1})$ is not a ring isomorphism. This unexpected result comes from the fact that the definitions of the Sbox and of the linear layer do not use the same representation: the Sbox is defined over \mathbf{F}_2^4 while the linear layer is defined over \mathbf{F}_{2^4} . This is why the choice of the basis affects the MEDP while this is obviously not the case when the two functions are defined over the same alphabet. But, even if this does not correspond to a natural mathematical description, it may be relevant to use the binary representation for the Sbox (chosen to minimize the number of gates for instance), while the field representation is used for the mixing layer since it is \mathbf{F}_{2^m} -linear (see e.g. [25]).

It is worth noticing that the previous situation is not related to the fact that one of the field representations is defined by a non-primitive polynomial. Indeed, the following example shows that even changing the primitive polynomial used for constructing \mathbf{F}_{2^m} may affect the two-round MEDP and MELP.

Example 3. We now consider two rounds of a cipher $\text{SPN}(5, 4, S, M)$ where S is the following permutation of \mathbf{F}_2^5 :

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	0	1	18	20	25	16	6	27	17	3	22	15	31	7	30	26
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$S(x)$	4	23	29	21	9	10	24	2	14	5	13	8	28	19	12	11

When \mathbf{F}_{2^5} is identified with \mathbf{F}_5 by the basis $\{1, \alpha, \alpha^2, \alpha^3, \alpha^4\}$ where α is a root of the primitive polynomial $X^5 + X^2 + 1$, Theorem 2 shows that any MDS diffusion layer linear over \mathbf{F}_{2^5} with this representation satisfies

$$\text{MEDP}_2^E \leq 13 \times 2^{-20} \text{ and } \text{MELP}_2^E \leq 8407 \times 2^{-27}.$$

When \mathbf{F}_{2^5} is constructed from the primitive polynomial $X^5 + X^3 + 1$, the lower and upper bounds from Th. 2 and Prop. 3 are equal, and show that there exists some MDS layer linear over \mathbf{F}_{2^5} with this alternative representation such that

$$\text{MEDP}_2^E = 14 \times 2^{-20} \text{ and } \text{MELP}_2^E = 8663 \times 2^{-27}.$$

The first primitive polynomial then guarantees lower two-round MEDP and MELP than the second one.

Another example is the LED block cipher. In [25, Section 3.2], the designers provide an upper bound on the four-round MEDP and MELP of a cipher of the form $\text{SPN}_F(4, 4, \mathcal{S}, \mathcal{M})$ ³ where \mathcal{M} is an \mathbf{F}_{2^4} -linear function with branch number 5, \mathcal{S} corresponds to the Present Sbox and \mathbf{F}_{2^4} is defined by the basis $(1, \alpha, \alpha^2, \alpha^3)$, with α a root of $X^4 + X + 1$. To this end, they use the FSE 2003 bound, which leads to $\text{MEDP}_2 \leq 2^{-8}$ and $\text{MELP}_2 \leq 2^{-8}$, implying that $\text{MEDP}_4 \leq 2^{-32}$ and $\text{MELP}_4 \leq 2^{-32}$. For this cipher, our new upper bound is equal to the FSE 2003 bound and then does not improve the result. However, if we consider the same Sbox, but modify the representation of \mathbf{F}_{2^4} and choose the basis defined by $X^4 + X^3 + 1$, Theorem 2 leads to $\text{MEDP}_2 \leq 3 \times 2^{-10}$. Then, with this minor modification, the upper bound on MEDP_4 is improved by a factor $(3/4)^4 = 0.3164$ (while the bound on MELP_4 is unchanged).

4 Multiplicative Invariance for Sboxes

Power permutations are often considered as suitable Sboxes since determining their differential and Walsh spectra is easier and also because they usually have a lower implementation cost. This family of Sboxes is also of great interest in our context since our bounds provide a very good approximation of the exact two-round MEDP and MELP which depends on the Sbox and on the branch number only. Indeed, for power permutations, we get a universal lower bound in the sense that the bound provided in Prop. 3 holds for any \mathbf{F}_{2^m} -linear permutation \mathcal{M} . This comes from the fact that all rows in the difference table (resp. in the correlation table) of a power permutation can be deduced from a single one. This is because any power function \mathcal{S} is an endomorphism over the multiplicative group $\mathbf{F}_{2^m}^*$, i.e., $\mathcal{S}(xy) = \mathcal{S}(x)\mathcal{S}(y)$ for any pair of nonzero elements (x, y) . Unfortunately, there is no hope to capture a larger family of Sboxes with a straightforward generalization of this property since it can be easily shown that any function \mathcal{S} satisfying $\mathcal{S}(xy) = \mathcal{S}(x)\mathcal{S}'(y)$ for some \mathcal{S}' is of the form $\mathcal{S}(x) = cx^s$. However, we can define this suitable multiplicative property on the difference table (resp. on the Walsh transform) of \mathcal{S} , and not on the function itself.

4.1 Generalizing the Multiplicative Property

Definition 7. Let \mathcal{S} be a mapping of \mathbf{F}_{2^m} .

- \mathcal{S} is said to have multiplicative-invariant derivatives if, for any $x \in \mathbf{F}_{2^m}^*$ there exists a permutation π_x of $\mathbf{F}_{2^m}^*$ such that

$$\delta_F(\alpha, xy) = \delta_F(\pi_x(\alpha), y), \quad \forall y \in \mathbf{F}_{2^m}^*.$$

³ As for Prøst, this result does not directly apply to LED since the round keys are inserted every four rounds only.

- \mathcal{S} is said to have a multiplicative-invariant Walsh transform if, for any $x \in \mathbf{F}_{2^m}^*$ there exists a permutation ψ_x of $\mathbf{F}_{2^m}^*$ such that

$$\mathcal{W}_F(\alpha, xy)^2 = \mathcal{W}_F(\psi_x(\alpha), y)^2, \quad \forall y \in \mathbf{F}_{2^m}^*.$$

These definitions include all functions resulting from the composition on the right of a power permutation with an \mathbf{F}_2 -linear permutation (cf. proof in Appendix B).

Proposition 4. *Let $\mathcal{S} = \mathcal{S}' \circ L$ where L is an \mathbf{F}_2 -linear permutation of \mathbf{F}_{2^m} and $\mathcal{S}' : x \mapsto x^s$ is a power permutation over \mathbf{F}_{2^m} . Then, both the derivatives of \mathcal{S} and its Walsh transform are multiplicative-invariant.*

It is worth noticing that the fact that a permutation has multiplicative-invariant derivatives (resp. Walsh transform) does not imply that a similar property holds for its inverse. In other words, Prop. 4 does not apply to the composition on the left of a power permutation with a linear permutation. The following proposition shows that the permutations with multiplicative-invariant derivatives (resp. Walsh transform) are not all affine equivalent to a power permutation.

Proposition 5. *Let m be an odd integer and \mathcal{S} be a quadratic permutation of \mathbf{F}_{2^m} with $\Delta(\mathcal{S}) = 2$ (aka APN permutation). Then, \mathcal{S} has multiplicative-invariant derivatives and \mathcal{S}^{-1} has a multiplicative-invariant Walsh transform.*

This result actually applies to a (possibly) more general class of permutations known as *crooked permutations*, which includes all quadratic APN permutations (see details in Appendix B). Prop. 5 applies for instance to the infinite family of APN permutations of degree 2

$$x \longmapsto x^{2^i+1} + ux^{2^j\frac{m}{3} + 2^{(3-j)\frac{m}{3}+i}} \text{ with } \gcd(i, m) = 1 \text{ and } j = im/3 \bmod 3$$

over \mathbf{F}_{2^m} , m odd, divisible by 3 and not by 9, which is not affine equivalent to a power mapping [12].

4.2 A Universal Lower Bound for Sboxes with Some Multiplicative Invariance

We now show that for Sboxes with multiplicative-invariant derivatives (resp. Walsh transform), the previously established bounds simplify.

Proposition 6. *Let \mathcal{S} be a permutation of \mathbf{F}_{2^m} such that either \mathcal{S} or \mathcal{S}^{-1} has multiplication-invariant derivatives (resp. Walsh transform). For any integers d and d^\perp , we define*

$$\mathcal{B}'_u(\mu) = \max_{\alpha, \beta \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \delta_F(\alpha, \gamma)^u \delta_F(\gamma + \mu, \beta)^{(d-u)}, \quad \text{with } 1 \leq u < d,$$

$$\mathcal{B}'_u^\perp(\mu) = \max_{\alpha, \beta \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \mathcal{W}_F(\alpha, \gamma)^{2u} \mathcal{W}_F(\gamma + \mu, \beta)^{2(d^\perp-u)}, \quad \text{with } 1 \leq u < d^\perp.$$

Then, for any u , we have

$$\mathcal{B}_u(0) = \mathcal{B}'_u(0) \text{ and } \max_{\mu \in \mathbf{F}_{2^m}^*} \mathcal{B}_u(\mu) = \max_{\mu \in \mathbf{F}_{2^m}^*} \mathcal{B}'_u(\mu)$$

$$\text{resp. } \mathcal{B}_u^\perp(0) = \mathcal{B}'_u^\perp(0) \text{ and } \max_{\mu \in \mathbf{F}_{2^m}^*} \mathcal{B}_u^\perp(\mu) = \max_{\mu \in \mathbf{F}_{2^m}^*} \mathcal{B}'_u^\perp(\mu),$$

where $\mathcal{B}_u(\mu)$ and $\mathcal{B}_u^\perp(\mu)$ are defined as in Theorem 2.

It follows that the upper bounds defined by Theorem 2 simplify to

$$\text{MEDP}_2^E \leq 2^{-md} \max_{1 \leq u < d} \max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}'_u(\mu) \text{ and } \text{MELP}_2^E \leq 2^{-2md^\perp} \max_{1 \leq u < d^\perp} \max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}'_u^\perp(\mu).$$

More interestingly, we now get some universal lower bound on MEDP_2 and MELP_2 , i.e., which hold for any diffusion layer with maximal branch number.

Theorem 3. *Let \mathcal{S} be a permutation of \mathbf{F}_{2^m} . Then, for any \mathbf{F}_{2^m} -linear diffusion layer \mathcal{M} over $(\mathbf{F}_{2^m})^t$ with maximal branch number $d = t + 1$, the MEDP_2 and MELP_2 of any block cipher E of the form $\text{SPN}_F(m, t, \mathcal{S}, \mathcal{M})$ satisfy the following.*

- If both \mathcal{S} and \mathcal{S}^{-1} have multiplicative-invariant derivatives, then

$$\text{MEDP}_2^E \geq 2^{-m(t+1)} \max_{1 \leq u < d} \mathcal{B}'_u(0);$$

- if both \mathcal{S} and \mathcal{S}^{-1} have a multiplicative-invariant Walsh transform, then

$$\text{MELP}_2^E \geq 2^{-2m(t+1)} \max_{1 \leq u < d} \mathcal{B}'_u^\perp(0);$$

- if \mathcal{S} has multiplicative-invariant derivatives (resp. Walsh transform), then

$$\text{MEDP}_2^E \geq 2^{-m(t+1)} \mathcal{B}'_t(0), \text{ resp. } \text{MELP}_2^E \geq 2^{-2m(t+1)} \mathcal{B}'_t^\perp(0).$$

- if \mathcal{S}^{-1} has multiplicative-invariant derivatives (resp. Walsh transform), then

$$\text{MEDP}_2^E \geq 2^{-m(t+1)} \mathcal{B}'_1(0), \text{ resp. } \text{MELP}_2^E \geq 2^{-2m(t+1)} \mathcal{B}'_1^\perp(0).$$

Let us focus on all permutations of \mathbf{F}_{2^8} of the same form as the AES Sbox: $\mathcal{S}(x) = A(x^{254})$, where A is an \mathbf{F}_2 -affine permutation of \mathbf{F}_{2^8} . Since \mathcal{S}^{-1} has multiplication-invariant derivatives and Walsh transform (cf. Prop. 4), we derive from Theorems 2 and 3, and Prop. 6 that, for $t = 4$,

$$2^{-40} \mathcal{B}'_1(0) \leq \text{MEDP}_2 \leq 2^{-40} \max_{1 \leq u \leq 4} \max_{\mu \in \mathbf{F}_{2^8}^*} \mathcal{B}'_u(\mu)$$

$$\text{and } 2^{-80} \mathcal{B}'_1^\perp(0) \leq \text{MELP}_2 \leq 2^{-80} \max_{1 \leq u \leq 4} \max_{\mu \in \mathbf{F}_{2^8}^*} \mathcal{B}'_u^\perp(\mu).$$

These bounds do not depend on the isomorphism between \mathbf{F}_2^8 and \mathbf{F}_{2^8} since their expressions do not involve any multiplication in \mathbf{F}_{2^8} , while this was not the case of the more general bound in Theorem 2. Then, we get the following results for different choices of the affine permutation A .

- For the affine function A used in the AES, $\text{SPN}_F(8, 4, \mathcal{S}, \mathcal{M})$ satisfies

$$53 \times 2^{-34} \leq \text{MEDP}_2 \leq 55.5 \times 2^{-34} \text{ and } 1.638 \times 2^{-28} \leq \text{MELP}_2 \leq 1.86 \times 2^{-28}$$

for any \mathbf{F}_{2^8} -linear MDS diffusion layer \mathcal{M} and any isomorphism between \mathbf{F}_{2^8} and \mathbf{F}_2^8 . The exact values for the diffusion layer used in the AES correspond to the lower bounds in both cases. But, we have exhibited in Prop. 3 an MDS diffusion layer for which $\text{MELP}_2 \geq 1.66 \times 2^{-28}$. Then, the choice of the MDS linear layer affects the value of MELP_2 within this interval.

- For the affine function A' used in SHARK [42] and SQUARE [16] which are two predecessors of the AES, $\text{SPN}_F(8, 4, \mathcal{S}, \mathcal{M})$ satisfies

$$53 \times 2^{-34} \leq \text{MEDP}_2 \leq 56 \times 2^{-34} \text{ and } 1.7169 \times 2^{-28} \leq \text{MELP}_2 \leq 1.9847 \times 2^{-28}$$

for any \mathbf{F}_{2^8} -linear MDS diffusion layer \mathcal{M} . Then, the affine function chosen in the AES Sbox offers a slightly better guarantee than the one chosen in SQUARE. Indeed, it is impossible with the SQUARE affine function to obtain a two-round MELP which is as small as the one of the AES. Note that the isomorphism between \mathbf{F}_2^8 and \mathbf{F}_{2^8} is different in SQUARE and in the AES [2].

- We have exhibited a linear permutation A'' of \mathbf{F}_2^8 for which the corresponding Sbox is such that $\text{SPN}_F(8, 4, \mathcal{S}, \mathcal{M})$ satisfies

$$\text{MEDP}_2 = 56 \times 2^{-34} \text{ and } 1.8354 \times 2^{-28} \leq \text{MELP}_2 \leq 1.8684 \times 2^{-28}$$

for any \mathbf{F}_{2^8} -linear MDS diffusion layer \mathcal{M} . Then, this Sbox always provides a higher two-round MEDP than the AES Sbox.

Even if we are not able to explicitly construct an affine permutation A which minimizes the values of MEDP_2 and MELP_2 , our results clearly simplify the task of the designer. Indeed, the affine permutation A and the diffusion layer M can be chosen separately since a very good estimate of MEDP_2 and MELP_2 is obtained independently of the diffusion layer. This is more efficient than computing these values for many pairs (A, M) .

4.3 Involutions with Some Multiplicative Invariance

A particular case of interest is when \mathcal{S} is an involution with multiplicative-invariant derivatives (or Walsh transform). Then, the lower bound in the previous theorem corresponds to the upper bound in Theorem 2, and both values are equal to the FSE 2003 bound.

Corollary 2. *Let \mathcal{S} be an involution of \mathbf{F}_{2^m} with multiplicative-invariant derivatives (resp. Walsh transform). Then, for any t and any \mathbf{F}_{2^m} -linear diffusion layer \mathcal{M} over $\mathbf{F}_{2^m}^t$ with branch number $t + 1$, any block cipher of the form $\text{SPN}_F(m, t, \mathcal{S}, \mathcal{M})$ satisfies*

$$\text{MEDP}_2^E = 2^{m(t+1)} \max_{\alpha \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \delta_F(\alpha, \gamma)^{t+1},$$

$$\text{resp. } \text{MELP}_2^E = 2^{2m(t+1)} \max_{\alpha \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \mathcal{W}_F(\alpha, \gamma)^{2(t+1)}.$$

The naive Sbox, *i.e.* the inversion in \mathbf{F}_{2^m} , satisfies all hypotheses of the previous corollary. The exact values of MEDP₂ and MELP₂ for an SPN combining the naive Sbox over \mathbf{F}_{2^m} and any \mathbf{F}_{2^m} -linear layer with maximal branch number are then always equal to the FSE 2003 bounds. For instance, for the two-round AES with the naive Sbox, we have MEDP₂ = 79×2^{-34} and MELP₂ = $48,193,409 \times 2^{-52}$, and this is independent of the \mathbf{F}_{2^8} -linear MDS layer. In particular, the exact MEDP₂ and MELP₂ do not depend on the field representation since Coro. 2 provides the same value for any basis. Also, this explains why, among all Sboxes in the same equivalence class, the naive Sbox is the one which leads after two rounds both to the highest MEDP and to the highest MELP for any \mathbf{F}_{2^m} -linear diffusion layer with maximal branch number. And this situation is independent of the size of the Sbox, and of the choice of the \mathbf{F}_{2^m} -linear MDS layer.

5 Conclusions

We have improved the general upper bounds on the two-round MEDP and MELP for a given Sbox over \mathbf{F}_2^m and any \mathbf{F}_{2^m} -linear diffusion layer with given branch numbers. One of the main properties of these new bounds is that they are not invariant under affine equivalence, and then they enable the designers to choose an appropriate Sbox within an equivalence class, independently of the diffusion layer. These bounds point out the importance of some interactions between the Sbox and its inverse. In particular, the involutions play a special role since there always exists some diffusion layer for which both MEDP₂ and MELP₂ achieve the highest possible value we can obtain for an Sbox in the same equivalence class. Also, we have shown that, for the Sboxes with multiplicative-invariant derivatives or Walsh transform, we can compute a lower bound on MEDP₂ and MELP₂ independently of the choice of the MDS diffusion layer. This result applies for instance to all Sboxes of the form $x \mapsto A(x^s)$, as in the AES. In particular, we have proved that, independently of the specifications of the MDS diffusion layer, the naive Sbox leads to the highest possible MEDP₂ and MELP₂. The exact MEDP₂ and MELP₂ may even vary with the basis used for defining \mathbf{F}_{2^m} . Our work then raises several open questions. We have shown that involitional power permutations are the weakest Sboxes in their equivalence class whatever MDS linear layer is chosen. For involutions which do not have any multiplicative-invariant property, this result holds but for some MDS layers only. Then, it would be interesting to determine whether this weakness is more general, and whether an involution is always the worst choice within an equivalent class. This issue is of practical interest since involitional Sboxes are a natural choice for minimizing the implementation overhead of decryption on top of encryption. Another open question is whether the use of an involitional Sbox, especially of the naive Sbox, introduces a similar weakness for a higher number of rounds, in the sense of the conjecture in [17]. The difficulty comes from the fact that, exactly as for the FSE 2003 bound, applying our upper bound twice successively requires the knowledge of the whole difference table of the superbox. Our new bound can then be combined with (1) only, to get a bound of the 4-round MEDP and MELP.

Acknowledgments. The authors would like to thank Daniel Augot, Matthieu Finiasz, María Naya Plasencia for valuable discussions, and the reviewers for their constructive comments.

References

1. Abdelraheem, M.A., Ågren, M., Beelen, P., Leander, G.: On the distribution of linear biases: three instructive examples. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 50–67. Springer, Heidelberg (2012)
2. Barreto, P.S.: Implementation of the SQUARE block cipher. <http://www.larc.usp.br/~pbarreto/sqjava21.zip>
3. Bending, T.D., Fon-Der-Flaass, D.: Crooked Functions, Bent Functions, and Distance Regular Graphs. Electr. J. Comb. **5** (1998)
4. Bierbrauer, J., Kyureghyan, G.M.: Crooked binomials. Designs, Codes and Cryptography **46**(3), 269–301 (2008)
5. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology, 3–72 (1991)
6. Biryukov, A., De Cannière, C., Braeken, A., Preneel, B.: A toolbox for cryptanalysis: linear and affine equivalence algorithms. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 33–50. Springer, Heidelberg (2003)
7. Blondeau, C., Bogdanov, A., Leander, G.: Bounds in shallows and in miseries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 204–221. Springer, Heidelberg (2013)
8. Blondeau, C., Nyberg, K.: New links between differential and linear cryptanalysis. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 388–404. Springer, Heidelberg (2013)
9. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE – A low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
10. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE - A Low-latency Block Cipher for Pervasive Computing Applications (Full version). IACR Cryptology ePrint Archive 529 (2012)
11. Brinkmann, M., Leander, G.: On the classification of APN functions up to dimension five. Designs, Codes and Cryptography **49**(1–3), 273–288 (2008)
12. Budaghyan, L., Carlet, C., Leander, G.: Two Classes of Quadratic APN Binomials Inequivalent to Power Functions. IEEE Transactions on Information Theory **54**(9), 4218–4229 (2008)
13. Canteaut, A., Charpin, P.: Decomposing bent functions. IEEE Transactions on Information Theory **49**(8), 2004–2019 (2003)
14. Chun, K., Kim, S., Lee, S., Sung, S.H., Yoon, S.: Differential and linear cryptanalysis for 2-round SPNs. Inf. Process. Lett. **87**(5), 277–282 (2003)
15. Daemen, J.: Cipher and hash function design strategies based on linear and differential cryptanalysis. Ph.D. thesis, K.U. Leuven (1995)
16. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher SQUARE. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
17. Daemen, J., Lamberger, M., Pramstaller, N., Rijmen, V., Vercauteren, F.: Computational aspects of the expected differential probability of 4-round AES and AES-like ciphers. Computing **85**(1–2), 85–104 (2009)

18. Daemen, J., Rijmen, V.: The wide trail design strategy. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 222–238. Springer, Heidelberg (2001)
19. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002)
20. Daemen, J., Rijmen, V.: Understanding two-round differentials in AES. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 78–94. Springer, Heidelberg (2006)
21. Daemen, J., Rijmen, V.: Probability distributions of correlation and differentials in block ciphers. *Journal of Mathematical Cryptology* **1**(3), 221–242 (2007)
22. Daemen, J., Rijmen, V.: New criteria for linear maps in AES-like ciphers. *Cryptography and Communications* **1**(1), 47–69 (2009)
23. Daemen, J., Rijmen, V.: Correlation analysis in $GF(2^n)$. In: Advanced Linear Cryptanalysis of Block and Stream Ciphers. Cryptology and information security, pp. 115–131. IOS Press (2011)
24. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: A new family of lightweight block ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012)
25. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
26. Hong, S.H., Lee, S.-J., Lim, J.-I., Sung, J., Cheon, D.H., Cho, I.: Provable security against differential and linear cryptanalysis for the SPN structure. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 273–283. Springer, Heidelberg (2001)
27. Kavun, E.B., Lauridsen, M.M., Leander, G., Rechberger, C., Schwabe, P., Yalçın, T.: Prøst v1.1. Submission to the CAESAR competition (2014). <http://proest.compute.dtu.dk/proestv11.pdf>
28. Keliher, L., Sui, J.: Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. *IET Information Security* **1**(2), 53–57 (2007)
29. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
30. Kyureghyan, G.M.: Crooked maps in \mathbf{F}_{2^n} . *Finite Fields and Their Applications* **13**(3), 713–726 (2007)
31. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Symposium on Communication, Coding and Cryptography. Kluwer Academic Publishers (1994)
32. Lai, X., Massey, J.L., Murphy, S.: Markov ciphers and differential cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991)
33. Leander, G.: On linear hulls, statistical saturation attacks, PRESENT and a cryptanalysis of PUFFIN. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 303–322. Springer, Heidelberg (2011)
34. Leander, G., Poschmann, A.: On the classification of 4 bit S-Boxes. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, pp. 159–176. Springer, Heidelberg (2007)
35. Lim, C.H., Korkishko, T.: mCrypton – A lightweight block cipher for security of low-cost RFID tags and sensors. In: Song, J.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2006)
36. MacWilliams, F.J., Sloane, N.J.: The theory of error-correcting codes. North-Holland (1977)
37. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)

38. Murphy, S.: The effectiveness of the linear hull effect. *J. Mathematical Cryptology* **6**(2), 137–147 (2012)
39. Nyberg, K.: Differentially uniform mappings for cryptography. In: Helleseth, T. (ed.) *EUROCRYPT 1993*. LNCS, vol. 765, pp. 55–64. Springer, Heidelberg (1994)
40. Nyberg, K.: Linear approximation of block ciphers. In: De Santis, A. (ed.) *EUROCRYPT 1994*. LNCS, vol. 950, pp. 439–444. Springer, Heidelberg (1995)
41. Park, S., Sung, S.H., Lee, S.-J., Lim, J.-I.: Improving the upper bound on the maximum differential and the maximum linear hull probability for SPN structures and AES. In: Johansson, T. (ed.) *FSE 2003*. LNCS, vol. 2887, pp. 247–260. Springer, Heidelberg (2003)
42. Rijmen, V., Daemen, J., Preneel, B., Bosselaers, A., Win, E.D.: The cipher SHARK. In: Gollmann, D. (ed.) *FSE 1996*. LNCS, vol. 1039, pp. 99–111. Springer, Heidelberg (1996)
43. Saarinen, M.-J.O.: Cryptographic analysis of all 4×4 -bit S-boxes. In: Miri, A., Vaudenay, S. (eds.) *SAC 2011*. LNCS, vol. 7118, pp. 118–133. Springer, Heidelberg (2012)
44. Tardy-Corfdir, A., Gilbert, H.: A known plaintext attack of FEAL-4 and FEAL-6. In: Feigenbaum, J. (ed.) *CRYPTO 1991*. LNCS, vol. 576, pp. 172–182. Springer, Heidelberg (1992)
45. Zheng, Y., Zhang, X.-M.: Plateaued functions. In: Varadharajan, V., Mu, Y. (eds.) *ICICS 1999*. LNCS, vol. 1726, pp. 284–300. Springer, Heidelberg (1999)

A Proofs of Theorem 2, Prop. 2 and 6, and Theorem 3

The new upper bounds on MEDP₂ and MELP₂ exploit the particular structure of the codewords in an \mathbf{F}_{2^m} -linear code, which is related to the notion of bundle introduced in [20]. In particular, we use the structure of the subsets of the code of the following form.

Definition 8. Consider a word c of length n and a subset $I \subseteq \{1, \dots, n\}$. The decomposition of c with respect to I is denoted by $(x, y)_I$: x corresponds to the restriction of c to I , and y corresponds to the restriction of c to the complement subset \bar{I} . For the sake of simplicity, the $|I|$ coordinates of x (resp. the coordinates of y) will be indexed by the elements of I (resp. of \bar{I}), i.e., $x_i = c_i$ for all $i \in I$ and $y_j = c_j$ for all $j \in \bar{I}$.

Lemma 2. Let \mathcal{C} be a linear code of length n , dimension k and minimum distance d over \mathbf{F}_{2^m} . For any subset $I \subset \{1, \dots, n\}$ of size $(n - d)$, and any $x \in (\mathbf{F}_{2^m})^{n-d}$, we define

$$Z(I, x) = \{y : (x, y)_I \in \mathcal{C}\}.$$

Then, for any I of size $(n - d)$,

- either $Z(I, 0)$ is empty or there exists some $y_0 \in (\mathbf{F}_{2^m})^d$ such that $Z(I, 0) = \{\gamma y_0, \gamma \in \mathbf{F}_{2^m}\}$;
- For any $x \neq 0$, either $Z(I, x)$ is empty or there exist some $y_0 \in (\mathbf{F}_{2^m})^d$ and some $y_1 \in (\mathbf{F}_{2^m})^d$ such that $Z(I, x) \subseteq \{y_1 + \gamma y_0, \gamma \in \mathbf{F}_{2^m}\}$.

Proof. – Assume that $Z(I, 0)$ is not empty. Since \mathcal{C} is \mathbf{F}_{2^m} -linear, for any $y_0 \in Z(I, 0)$, $(0, y_0)_I$ belongs to \mathcal{C} , implying that all $\gamma(0, y_0)_I$ with $\gamma \in \mathbf{F}_{2^m}$ belong to \mathcal{C} too.

- Let $x \neq 0$. Since the result obviously holds if $|Z(I, x)| \leq 1$, we suppose that $|Z(I, x)| \geq 2$. For any distinct y and y' in $Z(I, x)$, we get that both $c = (x, y)_I$ and $c' = (x, y')_I$ belong to \mathcal{C} , implying that $(y + y') \in Z(I, 0)$. From the previous result, there exists some y_0 such that $y + y' = \gamma y_0$ for some $\gamma \in \mathbf{F}_{2^m}$. It follows that y' is of the form $y' = y + \gamma y_0$. Since $wt(c + c') = wt(y + y')$ cannot be less than d , all coordinates of y_0 should be nonzero. \square

A.1 Proofs of Theorem 2 and Proposition 2

As in [41], we will use the following generalized version of Hölder inequality.

Lemma 3. [41, Lemma 1] Let $\{x_i^{(j)}\}_{i=1}^n$, $1 \leq j \leq p$, be p sequences of n real numbers. Then

$$\sum_{i=1}^n \left| \prod_{j=1}^p x_i^{(j)} \right| \leq \prod_{j=1}^p \left(\sum_{i=1}^n |x_i^{(j)}|^p \right)^{\frac{1}{p}}.$$

Now, we prove the following generic version of Theorem 2.

Theorem 4. Let m and t be two positive integers. Let Λ be a $2^m \times 2^m$ matrix with coefficients $\Lambda(\alpha, \beta), (\alpha, \beta) \in (\mathbf{F}_{2^m})^2$, in $[0; 1]$ such that $\Lambda(\alpha, 0) = \Lambda(0, \alpha) = 0$ for any $\alpha \neq 0$, and

$$\sum_{\beta \in \mathbf{F}_{2^m}} \Lambda(\alpha, \beta) = \sum_{\beta \in \mathbf{F}_{2^m}} \Lambda(\beta, \alpha) = 1, \text{ for all } \alpha \in \mathbf{F}_{2^m}.$$

Then, for any \mathbf{F}_{2^m} -linear code \mathcal{C} of length $(2t)$ with minimum distance d and for any nonzero a and b in $\mathbf{F}_{2^m}^t$, we have:

$$\Lambda_{a,b} = \sum_{c \in \mathcal{C}} \left(\prod_{i=1}^t \Lambda(a_i, c_i) \right) \left(\prod_{j=1}^t \Lambda(c_{t+j}, b_j) \right) \leq \max_{1 \leq u < d} \max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}_u(\mu)$$

$$\text{where } \mathcal{B}_u(\mu) = \max_{\alpha, \beta, \lambda \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma)^u \Lambda(\gamma \lambda + \mu, \beta)^{d-u}.$$

Proof. Let a, b be nonzero elements of $(\mathbf{F}_{2^m})^t$. For any codeword c such that $\text{Supp}(c) \neq \text{Supp}(a, b)$, there exists $\ell \in \{1, \dots, t\}$ such that $\Lambda(a_\ell, c_\ell) = 0$ or $\Lambda(c_{t+\ell}, b_\ell) = 0$. Then,

$$\Lambda_{a,b} = \sum_{c \in \mathcal{C}: \text{Supp}(c) = \text{Supp}(a,b)} \left(\prod_{i=1}^t \Lambda(a_i, c_i) \right) \left(\prod_{j=1}^t \Lambda(c_{t+j}, b_j) \right).$$

We assume that $wt(a) + wt(b) \geq d$, otherwise the value $\Lambda_{a,b}$ is equal to zero, as there is no $c \in \mathcal{C}$ such that $\text{Supp}(c) = \text{Supp}(a, b)$. Then we can choose a pair

of subsets I_1 and I_2 of $\{1, \dots, t\}$ such that $I_1 \subseteq \text{Supp}(a)$, $I_2 \subseteq \text{Supp}(b)$ and $|I_1| + |I_2| = d$. We decompose any codeword c whose support equals $\text{Supp}((a, b))$ into two parts: $c = (y, x)_I$ where $I = (\{1, \dots, t\} \setminus I_1) \cup \{t+j, j \notin I_2\}$. In other words, y corresponds to the restriction of c to the positions outside I_1 and I_2 , while x corresponds to the other d positions. Recall that, following Definition 8, the coordinates of y (resp. of x) are indexed by the elements of I (resp. of $I_1 \cup \{t+j, j \in I_2\}$). Then, for $Z(I, y) = \{x : (y, x)_I \in \mathcal{C}\}$,

$$\Lambda_{a,b} = \sum_{y \in \mathbf{F}_{2^m}^{n-d}} \left(\prod_{i \notin I_1} \Lambda(a_i, y_i) \right) \left(\prod_{j \notin I_2} \Lambda(y_{t+j}, b_j) \right) \mathcal{Q}_{a,b}(I, y) \quad (5)$$

$$\text{where } \mathcal{Q}_{a,b}(I, y) = \sum_{x \in Z(I, y)} \left(\prod_{i \in I_1} \Lambda(a_i, x_i) \right) \left(\prod_{j \in I_2} \Lambda(x_{t+j}, b_j) \right).$$

We aim at finding an upper bound on $\mathcal{Q}_{a,b}(I, y)$. Let $u = |I_1|$. From Lemma 3,

$$\begin{aligned} \mathcal{Q}_{a,b}(I, y) &= \sum_{x \in Z(I, y)} \prod_{i \in I_1} \left[\Lambda(a_i, x_i) \left(\prod_{j \in I_2} \Lambda(x_{t+j}, b_j) \right)^{\frac{1}{u}} \right] \\ &\leq \prod_{i \in I_1} \left[\sum_{x \in Z(I, y)} \Lambda(a_i, x_i)^u \left(\prod_{j \in I_2} \Lambda(x_{t+j}, b_j) \right)^{\frac{1}{u}} \right]. \end{aligned}$$

For any $i \in I_1$, we apply Lemma 3 again:

$$\begin{aligned} \sum_{x \in Z(I, y)} \Lambda(a_i, x_i)^u \left(\prod_{j \in I_2} \Lambda(x_{t+j}, b_j) \right) &= \sum_{x \in Z(I, y)} \prod_{j \in I_2} \left(\Lambda(a_i, x_i)^{\frac{u}{d-u}} \Lambda(x_{t+j}, b_j) \right) \\ &\leq \prod_{j \in I_2} \left(\sum_{x \in Z(I, y)} \Lambda(a_i, x_i)^u \Lambda(x_{t+j}, b_j)^{d-u} \right)^{\frac{1}{d-u}} \end{aligned}$$

Now, we know from Lemma 2 that, if $Z(I, y) \neq \emptyset$, there exist $\alpha \in (\mathbf{F}_{2^m}^*)^d$ and $\beta \in (\mathbf{F}_{2^m})^d$ such that $Z(I, y) \subseteq \{\gamma\alpha + \beta, \gamma \in \mathbf{F}_{2^m}\}$. Then, for any pair $(i, j) \in I_1 \times I_2$, we can write:

$$\begin{aligned} \sum_{x \in Z(I, y)} \Lambda(a_i, x_i)^u \Lambda(x_{t+j}, b_j)^{d-u} &\leq \sum_{\gamma \in \mathbf{F}_{2^m}} \Lambda(a_i, \gamma\alpha_i + \beta_i)^u \Lambda(\gamma\alpha_{t+j} + \beta_{t+j}, b_j)^{d-u} \\ &= \sum_{\gamma' \in \mathbf{F}_{2^m}^*} \Lambda(a_i, \gamma')^u \Lambda(\gamma'\lambda + \mu, b_j)^{d-u}, \end{aligned}$$

where the last equality is obtained by replacing $\gamma\alpha_i + \beta_i$ by γ' since $\alpha_i \neq 0$, and by setting $\lambda = \alpha_{t+j}\alpha_i^{-1}$ and $\mu = \beta_{t+j} + \alpha_{t+j}\alpha_i^{-1}\beta_i$. Moreover the sum can be

taken over all nonzero γ' since $\Lambda(a_i, \gamma') = 0$ for $\gamma' = 0$. Let

$$\mathcal{B}_u = \max_{a,b,\lambda \in \mathbf{F}_{2^m}^*} \max_{\mu \in \mathbf{F}_{2^m}} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(a, \gamma)^u \Lambda(\gamma \lambda + \mu, b)^{d-u}.$$

Then, we get

$$\mathcal{Q}_{a,b}(I, y) \leq \prod_{i \in I_1} \prod_{j \in I_2} \left(\sum_{x \in Z(I, y)} \Lambda(a_i, x_i)^u \Lambda(x_{t+j}, b_j)^{d-u} \right)^{\frac{1}{u(d-u)}} \leq \mathcal{B}_u^{\frac{u(d-u)}{u(d-u)}} = \mathcal{B}_u.$$

Using (5) and $\sum_{\beta \in \mathbf{F}_{2^m}} \Lambda(\alpha, \beta) = \sum_{\alpha \in \mathbf{F}_{2^m}} \Lambda(\alpha, \beta) = 1$, we eventually deduce

$$\begin{aligned} \Lambda_{a,b} &= \sum_{y \in \mathbf{F}_{2^m}^{n-d}} \left(\prod_{i \notin I_1} \Lambda(a_i, y_i) \right) \left(\prod_{j \notin I_2} \Lambda(y_{t+j}, b_j) \right) \mathcal{Q}_{a,b}(I, y) \\ &\leq \mathcal{B}_u \sum_{y \in \mathbf{F}_{2^m}^{n-d}} \left(\prod_{i \notin I_1} \Lambda(a_i, y_i) \right) \left(\prod_{j \notin I_2} \Lambda(y_{t+j}, b_j) \right) \leq \mathcal{B}_u. \end{aligned} \quad \square$$

Theorem 2 is derived by observing that, up to a constant factor, $\Lambda_{a,b} = \text{EDP}_2(a, \mathcal{M}(b))$ for $\Lambda(\alpha, \beta) = 2^{-m} \delta_F(\alpha, \beta)$ and \mathcal{C} is the code $\mathcal{C}_{\mathcal{M}}$ defined in Definition 5, while $\Lambda_{a,b} = \text{ELP}_2(a, (\mathcal{M}^*)^{-1}(b))$ for $\Lambda(\alpha, \beta) = 2^{-2m} \mathcal{W}_F(\alpha, \beta)^2$ when \mathcal{C} is the code $\mathcal{C}_{\mathcal{M}}^\perp$ and \mathcal{M}^* denotes the adjoint of \mathcal{M} .

In the same way, we now prove the following generic version of Prop. 2.

Proposition 7. *Let m and d be two positive integers and Λ be a $2^m \times 2^m$ matrix satisfying the same hypotheses as in Theorem 4. Then, for any $1 \leq u < d$ and any $\mu \in \mathbf{F}_{2^m}$, we have*

$$\mathcal{B}_u(\mu) \leq \max \left(\max_{a \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(a, \gamma)^d, \max_{b \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\gamma, b)^d \right).$$

Moreover, if $\Lambda(\alpha, \beta) = \Lambda(\beta, \alpha)$ for any $(\alpha, \beta) \in (\mathbf{F}_{2^m})^2$, we have that, for any $1 \leq u < d$,

$$\max_{\mu \in \mathbf{F}_{2^m}} \mathcal{B}_u(\mu) = \mathcal{B}_u(0) = \max_{a \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(a, \gamma)^d = \max_{b \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\gamma, b)^d.$$

Proof. Lemma 2 implies that, for any set of p sequences $\{x_i^{(j)}\}_{i=1}^n$, $1 \leq j \leq p$,

$$\sum_{i=1}^n \left| \prod_{j=1}^p x_i^{(j)} \right| \leq \max_{1 \leq j \leq p} \sum_{i=1}^n |x_i^{(j)}|^p.$$

Using this inequality with $p = d$, we get that, for any $1 \leq u < d$, $\alpha, \beta, \lambda \in \mathbf{F}_{2^m}^*$ and $\mu \in \mathbf{F}_{2^m}$

$$\sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma\lambda + \mu)^u \Lambda(\gamma, \beta)^{d-u} \leq \max \left(\sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma)^d, \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\gamma\lambda + \mu, \beta)^d \right).$$

Since $\lambda \neq 0$, we have $\sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\gamma\lambda + \mu, \beta)^d = \sum_{\gamma' \in \mathbf{F}_{2^m}^*} \Lambda(\gamma', \beta)^d$. The inequality then follows.

Now, we assume that $\Lambda(a, b) = \Lambda(b, a)$ for any pair (a, b) . Then,

$$\sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma)^u \Lambda(\gamma\lambda + \mu, \beta)^{d-u} = \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma)^u \Lambda(\beta, \gamma\lambda + \mu)^{d-u}.$$

For $\mu = 0$, the maximum of this value over all nonzero α, β, λ is then greater than or equal to the value obtained for $\beta = \alpha$ and $\lambda = 1$, implying that

$$\mathcal{B}_u(0) \geq \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma)^u \Lambda(\alpha, \gamma)^{d-u} = \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma)^d.$$

Then, $\max_{\alpha \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(a, \gamma)^d$ is a lower bound for $\mathcal{B}_u(0)$, and then for $\max_{\mu} \mathcal{B}_u(\mu)$. Since we have proved that it is also an upper bound, we conclude that both quantities are equal. \square

A.2 Proofs of Proposition 6 and of Theorem 3

We now prove that for any Sbox \mathcal{S} such that either \mathcal{S} or \mathcal{S}^{-1} has multiplicative-invariant derivatives (resp. Walsh transform), the bound defined in Theorem 2 simplifies as explained in Prop. 6. Again, we give a generic version of this proposition which captures both settings.

Proposition 8. *Let m and d be two positive integers and Λ be a $2^m \times 2^m$ matrix satisfying the same hypotheses as in Theorem 4. Let*

$$\mathcal{B}'_u(\mu) = \max_{\alpha, \beta \in \mathbf{F}_{2^m}^*} \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma)^u \Lambda(\gamma + \mu, \beta)^{(d-u)}, \text{ with } 1 \leq u < d.$$

Assume that one of the following two conditions holds:

- (i) for any $x \in \mathbf{F}_{2^m}^*$ there is a permutation π_x of $\mathbf{F}_{2^m}^*$ such that $\Lambda(\alpha, xy) = \Lambda(\pi_x(\alpha), y)$, $\forall y \in \mathbf{F}_{2^m}^*$;
- (ii) for any $x \in \mathbf{F}_{2^m}^*$ there is a permutation ψ_x of $\mathbf{F}_{2^m}^*$ such that $\Lambda(xy, \alpha) = \Lambda(y, \psi_x(\alpha))$, $\forall y \in \mathbf{F}_{2^m}^*$.

Then, the quantities $\mathcal{B}_u(\mu)$ defined in Theorem 4 satisfy

$$\mathcal{B}_u(0) = \mathcal{B}'_u(0) \text{ and } \max_{\mu \in \mathbf{F}_{2^m}^*} \mathcal{B}_u(\mu) = \max_{\mu \in \mathbf{F}_{2^m}^*} \mathcal{B}'_u(\mu).$$

Proof. If Condition (i) holds, we have for any $\alpha, \beta, \lambda \in \mathbf{F}_{2^m}^*$ and any $\mu \in \mathbf{F}_{2^m}$,

$$\begin{aligned}\mathcal{B}_u(\alpha, \beta, \lambda, \mu) &= \sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \gamma)^u \Lambda(\gamma \lambda + \mu, \beta)^{d-u} \\ &= \sum_{\gamma' \in \mathbf{F}_{2^m}^*} \Lambda(\alpha, \lambda^{-1}(\gamma' + \mu))^u \Lambda(\gamma', \beta)^{d-u} = \mathcal{B}_u(\pi_{\lambda^{-1}}(\alpha), \beta, 1, \mu).\end{aligned}$$

The result then follows. If (ii) holds, we get $\mathcal{B}_u(\alpha, \beta, \lambda, \mu) = \mathcal{B}_u(\alpha, \psi_\lambda(\beta), 1, \mu \lambda^{-1})$ in a similar way. \square

A generic version of Theorem 3 is then the following.

Theorem 5. Let m and t be 2 positive integers and Λ a $2^m \times 2^m$ matrix satisfying the hypotheses of Th. 4. Assume that one of the following holds:

- (i) for any $x \in \mathbf{F}_{2^m}^*$ there is a permutation π_x of $\mathbf{F}_{2^m}^*$ such that $\Lambda(\alpha, xy) = \Lambda(\pi_x(\alpha), y)$, $\forall y \in \mathbf{F}_{2^m}^*$;
- (ii) for any $x \in \mathbf{F}_{2^m}^*$ there is a permutation ψ_x of $\mathbf{F}_{2^m}^*$ such that $\Lambda(xy, \alpha) = \Lambda(y, \psi_x(\alpha))$, $\forall y \in \mathbf{F}_{2^m}^*$.

Let $M\Lambda$ be defined by

$$M\Lambda = \max_{a, b \neq 0} \sum_{c \in \mathcal{C}} \left(\prod_{i=1}^t \Lambda(a_i, c_i) \right) \left(\prod_{j=1}^t \Lambda(c_{t+j}, b_j) \right).$$

with \mathcal{C} any \mathbf{F}_{2^m} -linear code of length $2t$, dimension t and $d_{\min} = t + 1$. Then,

- If both (i) and (ii) hold, then $M\Lambda \geq \max_{1 \leq u < d} \mathcal{B}'_u(0)$.
- If (i) holds, then $M\Lambda \geq \mathcal{B}'_t(0)$.
- If (ii) holds, then $M\Lambda \geq \mathcal{B}'_1(0)$.

Proof. For any fixed u , $1 \leq u \leq t$, we consider $\widehat{\alpha}, \widehat{\beta} \in \mathbf{F}_{2^m}^*$ some values for which

$$\sum_{\gamma \in \mathbf{F}_{2^m}^*} \Lambda(\widehat{\alpha}, \gamma)^u \Lambda(\gamma, \widehat{\beta})^{(d-u)} = \mathcal{B}'_u(0).$$

Since \mathcal{C} is MDS, any set of $(t+1)$ positions is the support of a minimum-weight codeword [36, Page 319]. Let then $c \in \mathcal{C}$ with support $I = \{1, \dots, u\} \cup \{t+1, \dots, 2t+1-u\}$. From Lemma 2, we know that the codewords with support I are the elements $\gamma c, \gamma \in \mathbf{F}_{2^m}^*$. We now examine the 3 cases.

- If both (i) and (ii) hold, then for any pair (a, b) , we have

$$\begin{aligned}\Lambda_{a,b} &= \sum_{c \in \mathcal{C}} \left(\prod_{i=1}^t \Lambda(a_i, c_i) \right) \left(\prod_{j=1}^t \Lambda(c_{t+j}, b_j) \right) \\ &= \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\prod_{i=1}^t \Lambda(a_i, \gamma c_i) \right) \left(\prod_{j=1}^t \Lambda(\gamma c_{t+j}, b_j) \right) \\ &= \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\prod_{i=1}^t \Lambda(\pi_{c_i}(a_i), \gamma) \right) \left(\prod_{j=1}^t \Lambda(\gamma, \psi_{c_{t+j}}(b_j)) \right).\end{aligned}$$

We choose a and b as $a_i = \pi_{c_i}^{-1}(\widehat{\alpha})$ for $1 \leq i \leq u$, $a_i = 0$ otherwise, and $b_j = \psi_{c_{t+j}}^{-1}(\widehat{\beta})$ for $1 \leq j \leq t+1-u$, $b_j = 0$ otherwise. Then, for these values,

$$\Lambda_{a,b} = \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\prod_{i=1}^t \Lambda(\widehat{\alpha}, \gamma) \right) \left(\prod_{j=1}^{t+1-u} \Lambda(\gamma, \widehat{\beta}) \right) = \mathcal{B}'_u(0) .$$

Since such a pair (a, b) can be defined for any $1 \leq u < d$, we deduce that $M\Lambda \geq \max_{1 \leq u < d} \mathcal{B}'_u(0)$.

- If only (i) holds, then we consider $u = t$ and we define a and b by $a_i = \pi_{c_i c_{t+1}}^{-1}(\widehat{\alpha})$ for $1 \leq i \leq t$, $b_1 = \widehat{\beta}$ and $b_j = 0$ for $j > 1$. Then, we get

$$\begin{aligned} \Lambda_{a,b} &= \sum_{\gamma \in \mathbf{F}_{2^m}^*} \left(\prod_{i=1}^t \Lambda(a_i, \gamma c_i) \right) \Lambda(\gamma c_{t+1}, b_1) \\ &= \sum_{\gamma' \in \mathbf{F}_{2^m}^*} \left(\prod_{i=1}^t \Lambda(a_i, \gamma' c_i c_{t+1}^{-1}) \right) \Lambda(\gamma', b_1) \\ &= \sum_{\gamma' \in \mathbf{F}_{2^m}^*} \left(\prod_{i=1}^t \Lambda(\pi_{c_i c_{t+1}}^{-1}(a_i), \gamma') \right) \Lambda(\gamma', b_1) \\ &= \sum_{\gamma' \in \mathbf{F}_{2^m}^*} \Lambda(\widehat{\alpha}, \gamma')^t \Lambda(\gamma', \widehat{\beta}) = \mathcal{B}'_t(0) . \end{aligned}$$

- If only (ii) holds, then we choose $u = 1$ and define a and b by $a_1 = \widehat{\alpha}$, $a_i = 0$ for $i > 1$, and $b_j = \varphi_{c_{t+j} c_1}^{-1}(\widehat{\beta})$ for $1 \leq j \leq t$. Then we get that $\Lambda_{a,b} = \mathcal{B}'_1(0)$ \square

B Proofs of Propositions 4 and 5

We now prove that for any mapping $\mathcal{S} = \mathcal{S}' \circ A$ where A is an \mathbf{F}_2 -affine permutation of \mathbf{F}_{2^m} and $\mathcal{S}' : x \mapsto x^s$, both the derivatives of \mathcal{S} and its Walsh transform are multiplicative-invariant.

Proof. From Lemma 1, it is known that

$$\delta_F^{\mathcal{S}}(a, b) = \delta_F^{\mathcal{S}'}(L(a), b) \text{ and } \mathcal{W}_F^{\mathcal{S}}(a, b)^2 = \mathcal{W}_F^{\mathcal{S}'}((L^{-1})^*(a), b)^2 ,$$

where $L : x \mapsto A(x) + A(0)$. Since $\mathcal{S}'(x) = x^s$, we have

$$\begin{aligned} \delta_F^{\mathcal{S}'}(a, bc) &= \#\{x \in \mathbf{F}_{2^m}, (x+a)^s + x^s = bc\} \\ &= \#\{x \in \mathbf{F}_{2^m}, (c^{-e}x + c^{-e}a)^s + (c^{-e}x)^s = b\} = \delta_F^{\mathcal{S}'}(c^{-e}a, b) \end{aligned}$$

where $x \mapsto x^e$ is the compositional inverse of \mathcal{S}' , i.e., e is the inverse of s modulo $(2^m - 1)$, and

$$\mathcal{W}_F^{\mathcal{S}'}(a, bc) = \sum_{x \in \mathbf{F}_{2^m}} (-1)^{\text{Tr}(bcx^s + ax)} = \sum_{x \in \mathbf{F}_{2^m}} (-1)^{\text{Tr}(by^s + ac^{-e}y)} = \mathcal{W}_F^{\mathcal{S}'}(c^{-e}a, b) .$$

Therefore, it follows that

$$\delta_F^S(a, bc) = \delta_F^{S'}(c^{-e}L(a), b) = \delta_F^S(\pi_c(a), b) \text{ with } \pi_c(a) = L^{-1}(c^{-e}L(a)) ,$$

$$\text{and } \mathcal{W}_F^S(a, bc) = \mathcal{W}_F^{S'}(c^{-e}(L^{-1})^*(a), b)^2 = \mathcal{W}_F^S(\psi_c(a), b)$$

with $\psi_c(a) = L^*(c^{-e}(L^{-1})^*(a))$, since $(L^{-1})^* = (L^*)^{-1}$. Clearly, both π_c and ψ_c are permutations for any nonzero c . \square

Now, we prove a generalized version of Prop. 5, which applies to a (possibly) larger family of mappings named crooked permutations.

Definition 9. [3] A function S from \mathbf{F}_{2^m} into \mathbf{F}_{2^m} is said to be *crooked* if, for any nonzero $\alpha \in \mathbf{F}_{2^m}$, $\text{Im}(D_\alpha S)$ is a linear or affine subspace of codimension 1, where $D_\alpha S : x \mapsto S(x + \alpha) + S(x)$.

It is known that all crooked permutations are APN and almost bent [3], and exist for m odd only. Clearly, any quadratic APN permutation is crooked. And it is highly conjectured that the crooked functions exactly correspond to the quadratic APN functions. This has been proved in [30] in the case of monomial functions and in [4] in the case of binomials. Now we can prove the following.

Proposition 9. Let S be a crooked permutation. Then, S has multiplicative-invariant derivatives and S^{-1} has a multiplicative-invariant Walsh transform.

Proof. Since S is a permutation, for any nonzero a , $D_a S$ cannot vanish implying that $\text{Im}(D_a S)$ is an affine hyperplane. Moreover, it is known that the $(2^m - 1)$ affine hyperplanes corresponding to $\text{Im}(D_a S)$ for all $a \neq 0$ are distinct [13, Lemma 5]. Therefore, there exists a permutation φ of \mathbf{F}_{2^m} with $\varphi(0) = 0$ such that $\text{Im}(D_a S) = \mathbf{F}_{2^m} \setminus \langle \varphi(a) \rangle^\perp$ for any nonzero a . Moreover, it is known (see e.g. [8]) that, for $u, v \in \mathbf{F}_{2^m}^*$,

$$\mathcal{W}_F^2(u, v) = \sum_{a, b \in \mathbf{F}_{2^m}} (-1)^{\text{Tr}(au+bv)} \delta_F(a, b) = 2^m + \sum_{a, b \in \mathbf{F}_{2^m}, a \neq 0} (-1)^{\text{Tr}(au+bv)} \delta_F(a, b).$$

The differential spectrum of S is determined by φ : for any $a \neq 0$, $\delta_F(a, b) = 1 - (-1)^{\text{Tr}(\varphi(a)b)}$. Then, for any $v \neq 0$, we get

$$\begin{aligned} \mathcal{W}_F^2(u, v) &= 2^m + \sum_{a, b \in \mathbf{F}_{2^m}, a \neq 0} (-1)^{\text{Tr}(au+bv)} - \sum_{a, b \in \mathbf{F}_{2^m}, a \neq 0} (-1)^{\text{Tr}(au+bv+\varphi(a)b)} \\ &= 2^m - \sum_{a \in \mathbf{F}_{2^m}, a \neq 0} (-1)^{\text{Tr}(au)} \left(\sum_{b \in \mathbf{F}_{2^m}} (-1)^{\text{Tr}(b(v+\varphi(a)))} \right) \\ &= 2^m - 2^m (-1)^{\text{Tr}(u\varphi^{-1}(v))} \end{aligned}$$

where the last equality uses the fact that $\varphi^{-1}(v) \neq 0$ when $v \neq 0$. It follows that

$$\mathcal{W}_F^2(xy, v) = 2^m - 2^m (-1)^{\text{Tr}(xy\varphi^{-1}(v))} = 2^m - 2^m (-1)^{\text{Tr}(y\varphi^{-1}(\pi_x(v)))}$$

where $\pi_x(v) = \varphi(x\varphi^{-1}(v))$. Moreover, for any nonzero x , π_x is a permutation. \square

Random Number Generators

A Provable-Security Analysis of Intel’s Secure Key RNG

Thomas Shrimpton^(✉) and R. Seth Terashima

Department of Computer Science, Portland State University, Portland, USA
`{teshrim,seth}@cs.pdx.edu`

Abstract. We provide the first provable-security analysis of the Intel Secure Key hardware RNG (ISK-RNG), versions of which have appeared in Intel processors since late 2011. To model the ISK-RNG, we generalize the PRNG-with-inputs primitive, introduced by Dodis et al. at CCS’13 for their `/dev/[u]random` analysis. The concrete security bounds we uncover tell a mixed story. We find that ISK-RNG lacks backward-security altogether, and that the forward-security bound for the “truly random” bits fetched by the `RDSEED` instruction is potentially worrisome. On the other hand, we are able to prove stronger forward-security bounds for the pseudorandom bits fetched by the `RDRAND` instruction. En route to these results, our main technical efforts focus on the way in which ISK-RNG employs CBCMAC as an entropy extractor.

Keywords: Random number generator · Entropy extraction · Provable security

1 Introduction

In late 2011, Intel began production of Ivy Bridge processors, which introduced a new pseudorandom number generator (PRNG), fully implemented in hardware. Access to this PRNG is through the `RDRAND` instruction (pronounced “read rand”), and benchmarks demonstrate a throughput of over 500 MB/s on a quad-core Ivy Bridge processor [10]. The forthcoming Broadwell architecture will support an additional instruction, `RDSEED` (“read seed”), which delivers true random bits, as opposed to cryptographically pseudorandom ones. Both `RDRAND` and `RDSEED` fall under the Intel Secure Key umbrella, so we will refer to the new hardware as the ISK-RNG [11].

The ISK-RNG has received a third-party lab evaluation [8], commissioned by Intel, but has yet to receive an academic, provable-security treatment along the lines of that given the `/dev/[u]random` software RNGs by a line of papers [1, 5, 7, 13]. We provide such a treatment.

Our abstract model for the ISK-RNG is that of a PRNG-with-input (PWI), established by Barak and Halevi [1] and extended by Dodis et al. [5]. To better capture important design features of the ISK-RNG we make several improvements to the PWI abstraction, which have significant knock-on effects for the

associated security notions. Our results establish the security of the ISK-RNG relative to these notions. Our findings are mixed, suggesting that in some cases RDSEED may not be as secure as one might hope, but with stronger results for RDRAND.

The ISK-RNG architecture. A detailed description of the ISK-RNG can be found in Section 3, but we'll provide a short sketch here. At a high-level, the ISK-RNG consists of four main components, as shown in Figure 1. At the heart is the hardware *entropy source*, which uses thermal noise to generate random bits and then writes them into a 256-bit raw-sample buffer. This buffer is subjected to a battery of heuristic *health tests*, which try to determine if the buffer contents are sufficiently random. The raw entropy bits are not assumed to be *uniformly* random—they may be biased or correlated. So a *conditioner* (i.e. an entropy extractor), repeatedly reads from this buffer, combining multiple 256-bit samples and compressing them into a single 128-bit string, hopefully one that is close to uniformly random.

These uniform bit strings then periodically reseed a deterministic PRNG (based on CTR-AES), providing a high-speed source of pseudorandom bits. Calls to the `RDRAND` instruction read from these bits, whereas calls to `RDSEED` will read directly from the conditioner output.

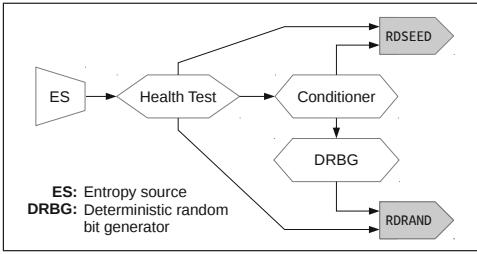


Fig. 1. Overview of the ISK-RNG

1.1 Security Findings for the ISK-RNG

We consider security of the ISK-RNG relative to four PWI-security notions, adopted (with modifications) from Dodis, Pointcheval, Ruhault, Vergnaud and Wichs [5] (hereafter DPRVW): *resilience*, the apparent randomness of `RDRAND` and `RDSEED` outputs; *forward security*, the apparent randomness of previous `RDRAND` and `RDSEED` outputs once the PWI state is revealed; *backward security*, the apparent randomness of future `RDRAND` and `RDSEED` outputs from a corrupted PWI state; and *robustness*, the apparent randomness of `RDRAND` and `RDSEED` outputs when state observation and corruption may happen at arbitrary times.

Using estimates for the quality of the entropy source derived from the findings of [8], we are able to show the following results (in a random permutation model):

1. As far as the resilience of `RDRAND` and `RDSEED` is concerned, `RDRAND` delivers pseudorandom bits with a comfortable security margin. On the other hand, `RDSEED` delivers truly random bits but with a security margin that

becomes worrisome if an adversary can see a large number of outputs from either interface. If he controls an unprivileged process on the same physical machine, this could happen very quickly.

2. For forward security, `RDRAND` and `RDSEED` also provide these respective security margins, as long as one is willing to make some reasonable assumptions about the adversary’s limitations.
3. The ISK-RNG does *not* provide backward security because the hardware indefinitely retains stale state when the ISK-RNG is not in active use. However, we are able to quantify the lifespan of this information when the ISK-RNG *is* in active use, thus proving backwards security and a read-only form of robustness against a class of “slow” adversaries.

Interpretation. In this context, forward security, backward security, and robustness are only relevant to those concerned about attackers who (1) are able to obtain physical access to the machine and (2) sophisticated enough to read or tamper with registers directly (the registers in question are not accessible through software, even by the operating system). Moreover, the window of opportunity for an attacker trying to compromise forward security (i.e., trying to reconstruct past random values given current access to the machine) is under a millisecond, barring pathological failures of the entropy source. Hence we suspect most practitioners will be concerned only with resilience.

As far as resilience, then, we prove `RDRAND` to be secure under a reasonable set of assumptions regarding the quality of the entropy source and a reasonable but heuristic assumption regarding AES-128: namely that it can be modeled as a random permutation when used with a specific fixed, publicly known key. We provide concrete, quantitative analysis in Section 7.3; the results are encouraging.

The situation with `RDSEED` is more complicated, because the security bounds become quantitatively quite weak in this context. We believe, but cannot prove, that this weakness does not correspond to a practical attack. Our suspicion is that an actual attack would require the adversary to have a precise physical model of the entropy source (the exact parameters of which appear to change from chip to chip [8]), and compute, by brute force, the distribution induced by processing streams from this entropy source using CBC-MAC under the previously mentioned AES key. Such an attack would clearly be computationally infeasible as long as the number of possible streams is large, but the relevant portion of the security bound is for computationally unbounded adversaries. (Recall that `RDSEED` is designed to provide truly random bits, rather than “merely” cryptographically pseudorandom ones.)

The stronger `RDRAND` results hold even if an attacker can access both interfaces.

Analyzing the ISK-RNG Entropy Extractor. The core technical results of the paper are concerned with analyzing the ISK-RNG entropy extractor, which employs CBC-MAC over AES-128, using the fixed string $\text{AES}_0(1)$ as the AES key. Although Intel documents [17] appeal to a CRYPTO’02 paper by Dodis, Gennaro, Håstad and Krawczyk [4] for support, this direct appeal is not well

founded. There are significant technical obstacles to overcome before these CBC-MAC results can be applied. For example, because extractor-dependent state is maintained across extractions (including state revealed to the adversary by RDSEED), a crucial “seed independence” assumption is violated. The CRI report [8], on the other hand, ignores the issue entirely by making an implicit assumption that applying CBC-MAC-AES to an arbitrary input with 128 bits of min-entropy will produce an output close to a uniformly random 128-bit string, an assumption known to be false with respect to *any* entropy extractor (not just CBC-MAC) [15]. We discuss and resolve these issues in Section 4.

1.2 Improvements to the PWI Model

For our abstract model, we take the *pseudorandom number generator with input* (PWI) primitive, formalized by DPRVW as a model for `/dev/[u]random`. At a high level, a PWI surfaces three algorithms: one to initialize the internal state of the primitive, one that produces an output for use by calling applications (updating the state in the process), and one that updates the state as a function of an *externally* provided input. Exposing an external input captures the practical situation in which PRNG outputs may depend upon external sources of (assumed) entropy.

One contribution of this paper is to generalize the PWI abstraction in ways that better capture not only the ISK-RNG, but also, we hope, other real-world PWIs. These include allowances for: non-uniform state, as is common in real-world PRNGs; realistic modeling of state setup procedures such as those in ISK-RNG¹; multiple external interfaces to the underlying state (e.g. RDRAND and RDSEED, as well as `/dev/[u]random`); and blocking behaviors.

To deal with non-uniform state, we introduce an analytical tool called a *masking function*. Loosely speaking, a masking function M is a tool for specifying what the “ideal” version $M(S)$ of any given PWI state S would be. This allows us to give general results about PWI security (e.g. what can be achieved when the state is ideal), yet admits per-scheme specification of what “ideal” means. We define masking functions, and incorporate them into the DPRVW’s security notions in such a way that their results can be quickly lifted to our setting. Masking functions also allow us to frame an appropriate definition for secure initialization: i.e. does the setup procedure produce a state S that is indistinguishable from $M(S)$?

2 Preliminaries

Notation. We denote the set of all n -bit strings as $\{0, 1\}^n$, and the set of all (finite) binary strings as $\{0, 1\}^*$. Given $x, y \in \{0, 1\}^*$, both xy and $x \parallel y$ denote their concatenation, and $|x|$ is the length of x . If $|x| = |y|$, $x \oplus y$ is the bitwise XOR of x and y . The symbol ε denotes the empty string. The set $\text{Perm}(n)$ denotes the set of permutations on $\{0, 1\}^n$.

¹ See [6, 9] for examples of what can go wrong when state initialization is weak.

When S is a finite set, we assume that it is equipped with the uniform distribution unless otherwise specified. For any distribution \mathcal{S} , the notation $X \xleftarrow{\$} \mathcal{S}$ indicates X is a random variable sampled from \mathcal{S} . Similarly, if F is a randomized algorithm, $X \xleftarrow{\$} F(x_1, \dots, x_n)$ means that X is sampled from the distribution induced by providing F with the indicated arguments. An adversary A is a randomized algorithm, and we adopt the shorthand $A \Rightarrow y$ to mean that when its execution halts, it outputs y . When an algorithm P is provided oracle (black-box, unit-time) access to an algorithm Q , we write P^Q .

Entropy and Sources. If X and X' are random variables, their statistical distance is $\Delta(X, X') = \frac{1}{2} \sum_x |\Pr[X = x] - \Pr[X' = x]|$, where the sum is over the union of the supports of X and X' . The min-entropy of X is $\mathbf{H}_\infty(X) = -\max_x (\log \Pr[X = x])$, and the worst-case min-entropy of X given X' is $\mathbf{H}_\infty(X | X') = -\log(\max_{x, x'} \Pr[X = x | X' = x'])$. When X is a random variable and \mathcal{E} is some event, we denote by $X|_{\mathcal{E}}$ the random variable X conditioned on \mathcal{E} ; i.e., for any x in the support of X , $\Pr[X|_{\mathcal{E}} = x] = \Pr[X = x | \mathcal{E}]$.

An *entropy source* \mathcal{D} is a randomized algorithm that, given a state string $\sigma \in \{0, 1\}^*$, samples a tuple $(\sigma', I, \gamma, z) \in \{0, 1\}^* \times \{0, 1\}^p \times \mathbb{R}_{\geq 0} \times \{0, 1\}^*$. Let $(\sigma_i, I_i, \gamma_i, z_i) \xleftarrow{\$} \mathcal{D}(\sigma_{i-1})$ be a sequence of samples, where $\sigma_0 = \varepsilon$, and $i = 1, \dots, q_{\mathcal{D}}$ for some integer $q_{\mathcal{D}}$. We say that entropy source \mathcal{D} is *legitimate* if $H_\infty(I_j | (I_i, z_i, \gamma_i)_{i \neq j}) \geq \gamma_j$. In this paper, we assume all entropy sources are legitimate.

In this definition, $\sigma, \sigma' \in \{0, 1\}^*$ represent the current and new states for \mathcal{D} , respectively. The string $I \in \{0, 1\}^p$ is what will be to be fed as input to the PWI, and should provide fresh entropy. The quantity $\gamma \in \mathbb{R}_{\geq 0}$ is an estimate for the amount of entropy contained in I . We note that γ is strictly a convenient book-keeping device in the PWI model, and is not intended to reflect an actual output of the entropy source being modeled. Our security notions will formalize attacker capabilities of interest, but we also allow for side-information (about I) that an attacker might obtain through means not otherwise explicit in the model (e.g. timing or power side-channels). This side information will be encoded in the string z .

Cryptographic Building Blocks. A blockcipher is a function $E : \{0, 1\}^\kappa \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that for each key $K \in \{0, 1\}^\kappa$, $E(K, \cdot)$, written $E_K(\cdot)$, is a permutation on $\{0, 1\}^n$. Given $\text{IV} \in \{0, 1\}^n$, $K \in \{0, 1\}^\kappa$, and $X_i \in \{0, 1\}^n$ for $i \in [0..n]$, define

$$\text{CTR}_K^{\text{IV}}(X_0 \cdots X_n) = (X_0 \oplus E_K(\text{IV})) \parallel \cdots \parallel (X_n \oplus E_K(\text{IV} + n)).$$

(We define the $+$ operator on $\{0, 1\}^n$ as addition modulo 2^n on the unsigned integers encoded by the operands.) Further define

$$\text{CBCMAC}_K^{\text{IV}}(X_0 \cdots X_n) = \text{CBCMAC}_K^{E_K(\text{IV} \oplus X_0)}(X_1 \cdots X_n),$$

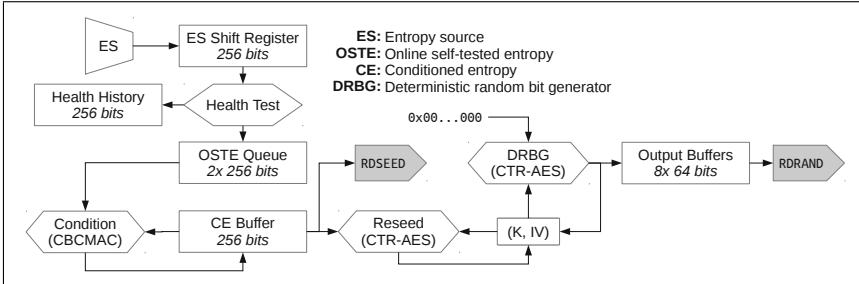


Fig. 2. Block diagram for Intel’s RDRAND implementation. The CBCMAC computation uses AES-128 with the fixed key $K' = \text{AES}_0(1)$. The DRBG runs AES-128 in counter mode to produce $\{0, 1\}^{128 \cdot 3}$ bits of output; the first 256 bits are used to update the key K and IV; the final 128 bits are sent to the output buffer, which is read by the RDRAND instruction.

and $\text{CBCMAC}_K^{\text{IV}}(\varepsilon) = \text{IV}$. Describing the standard CBCMAC algorithm in this manner simplifies descriptions of programs that compute CBCMAC online. We omit an explicit IV from the notation when $\text{IV} = 0^n$. In this paper, the implicit blockcipher E will always be AES-128 ($\kappa = n = 128$).

The pseudorandom-permutation (PRP) advantage of an adversary A attacking a blockcipher $E : \{0, 1\}^\kappa \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ is defined as $\text{Adv}_E^{\text{prp}}(A) = \Pr[A^{E_K} \Rightarrow 1] - \Pr[A^\pi \Rightarrow 1]$, with probabilities over the coins of A and the random variables $K \xleftarrow{\$} \{0, 1\}^\kappa$ and $\pi \xleftarrow{\$} \text{Perm}(n)$.

3 The ISK-RNG Architecture

This section describes the design of the ISK-RNG. Unless otherwise noted, this information comes from the CRI report [8]. The design can be divided roughly into three phases: entropy generation, entropy extraction, and expansion. Raw bits from the generation phase are fed into an entropy extractor, which is tasked with turning biased or correlated bits into uniform random strings. The expansion step uses these strings to seed a deterministic PRNG, which can produce cryptographically pseudorandom outputs at high speeds.

The design is shown in Figure 2. In this figure, rectangular boxes indicate values we consider part of the ISK-RNG state, hexagons indicate procedures that read and modify the state, and the shaded arrows indicate assembly instructions that allow (unprivileged) processes to read from the indicated buffer.

Entropy Generation, Health Tests, and “Swellness”. The hardware entropy source (labeled ES) is a dual differential jamb latch with feedback; thermal noise resolves a latch formed by two cross-coupled inverters, generating a random bit before the system is reset. Bits from the entropy source are written into a 256-bit shift register.

Every 256 writes, the contents of the register are subjected to a series of health tests. These count how many times certain specified bit strings appear, and verify that the results are within normal limits. For example, the substring

010 may occur between 9 and 57 times, inclusive. These substrings and the corresponding numbers of allowable occurrences are intended to catch pathologically bad failures while keeping the false-positive rate low. (For reference, a uniformly random 256-bit string would be flagged as unhealthy approximately 1% of the time.) If the current ES register fails one of the tests, that 256-bit source-sample is flagged as *unhealthy*. We refer interested readers to the CRI report [8] for a more detailed description; for our purposes, it suffices to say there is some fixed set $\mathcal{H} \subseteq \{0, 1\}^{256}$ of strings that pass the health tests. The health-history register tracks how many of the last 256 samples passed the health test. This is a first-in first-out buffer, where a 1-bit means that a sample was deemed healthy, and a 0-bit mean that a sample was deemed unhealthy. The global health of the ISK-RNG is captured by a property call *swellness*.

Definition 1 (Swell ISK-RNG). *The ISK-RNG is said to be swell if at least 128 of the last 256 samples were healthy, i.e. if the health-history register contains at least 128 1s.* \square

Whether or not the current sample passes the health test, it is appended to the Online Self-Tested Entropy (OSTE) queue, and it is the OSTE queue that provides input to the extraction phase.

Extraction. Strings in the OSTE queue are not assumed to be uniformly random. Instead, each 256-bit entry is assumed to have a certain amount of min-entropy. The CBCMAC construction, over AES with key $K' = \text{AES}_0(1)$ [12], is employed as an entropy extractor, in order to turn strings in the OSTE queue into two 128-bit *conditioned entropy* (CE) strings. These are held in the CE buffer, which is initially all zeros, and are used to service RDSEED instructions and to reseed the DRBG. An important property of the CE buffer is its *availability*.

Definition 2 (CE buffer availability). *The CE buffer is available if (1) the ISK-RNG is swell, and (2) both 128-bit halves of the CE buffer (CE_0 and CE_1) have been updated using m healthy OSTE values since the most recent RDSEED call and the most recent DRBG reseeding. For Ivy Bridge chips, $m = 2$; for Broadwell chips, $m = 3$ [12].* \square

When the CE buffer is not available, the hardware will replenish the OSTE buffers with fresh entropy and feed them into a running CBCMAC calculation until a sufficient number of healthy samples have been conditioned. So if at some point $\text{CE}_0 = X$ and then the CE buffer is used to service a RDSEED instruction (making the CE buffer unavailable), the hardware will collect entropy strings $I_1, I_2, I_3, \dots \in \{0, 1\}^{256}$ and reassign $\text{CE}_0 \leftarrow \text{CBCMAC}_0(XI_1I_2I_3\dots)$ online until there exist $i_1 < i_2 < \dots < i_m$ such that $I_{i_j} \in \mathcal{H}$ for $j \in [1..m]$ and the ISK-RNG is swell. Then the processes will repeat for CE_1 .

The particulars of the way CBCMAC is used in the ISK-RNG extractor, along with the notions of swellness and availability, play a large role in Section 4.

Expansion. To reseed the DRBG, the contents of CE_0 and CE_1 are used to generate a key and IV (respectively) for counter mode encryption over AES. This reseeding process only happens when the CE buffer is available. It takes the current key and IV, (K, IV) , and updates them by computing $K \parallel \text{IV} \leftarrow \text{CTR}_K^{\text{IV}}(\text{CE}_1 \parallel \text{CE}_2)$. Initially, $K = \text{IV} = 0^{128}$. However, using CTR with this non-random key is not a problem as long as the CE buffer is (close to) uniformly random: since the CE buffer is XORed into the CTR keystream, it can act as a one-time pad.

A pseudorandom value R is generated by computing $R \parallel K \parallel \text{IV} \leftarrow \text{CTR}_K^{\text{IV}}(0^{3 \cdot 128})$. (Note that this process also irreversibly updates K and IV , which helps provide forward security.) The ISK-RNG writes R to an output buffer, which is read by RDRAND. This FIFO output buffer [10] can contain up to eight 64-bit values. ISK-RNG allows a maximum of 511 64-bit values to be generated between reseeding operations; after this, it will only return 0s and will clear the carry bit to signal an error.

Setup. When the ISK-RNG powers on, the ISK-RNG performs a series of known-answer, built-in self-tests. Then the conditioned entropy (CE) buffer is cleared and the deterministic random bit generator (DRBG) is reseeded four times [12]. Each reseeding operation requires reconditioning the CE buffer until it is available. Finally, the system populates the eight output buffers using the DRBG.

Standards Compliance. Intel states [14] that ISK-RNG is compliant with NIST’s SP800-90B & C draft standards. Whereas RDRAND can provide bit strings with “only” a 128-bit security level (since it uses AES-128 in CTR mode), RDSEED has no such limitation.

4 Analysis of the ISK-RNG Extractor

As we will see, some of the PWI-security results for the ISK-RNG are not as strong as one might hope. Much of this is due to weak concrete bounds on its CBCMAC entropy extractor, which is tasked with turning the presumably biased and correlated bits from the entropy source into uniformly distributed strings. Let us explain.

Previous CBC-MAC Results Are Not (directly) Helpful. A paper by Dodis, Genarro, Håstad, Krawczyk and Rabin [4] analyzes the security of CBC-MAC as an entropy extractor, and their results are cited by Intel documents [17] to support the ISK-RNG design. Because generic PRFs-as-entropy-extractors results [3] are too weak to be useful, the analysis of [4] takes place in the random permutation model. That is, instead of considering CBC-MAC over a blockcipher with a random key, they consider CBC-MAC over a random permutation. This model is a heuristic: even, say, AES equipped with a random key would not be a random permutation. In fact, CBC-MAC within the ISK-RNG uses AES with the fixed key $K' = \text{AES}_0(1)$ (on all chips). This fact may strike one as alarming.

But we believe that a “nothing-up-my-sleeve” value for the extractor seed is a reasonable choice. (Generating the seed from the entropy source would be highly suspect from a theoretical perspective, because one requires that the extractor seed be “independent” of the entropy distribution.)

Anyway, our primary goal here is to identify what we *can* say about ISK, even if we’re forced to use a heuristic model. Dodis et al.[4] provide the following theorem:

Theorem 1 (CBCMAC entropy extractor [4]). *Fix positive integers k and L . Let $I \in \{0, 1\}^{Lk}$ be a random variable, $R \xleftarrow{\$} \{0, 1\}^k$ be a uniform random string, and let $\pi \xleftarrow{\$} \text{Perm}(k)$ be a random permutation. Then $\Delta((\pi, R), (\pi, \text{CBCMAC}_\pi(I))) \leq \frac{1}{2} \sqrt{2^{k-\mathbf{H}_\infty(I)} + \frac{\mathcal{O}(L^2)}{2^k}}$.*

Unfortunately, one cannot simply apply this theorem to the CBC-MAC-based extractor used in ISK-RNG, without attending to the following two significant obstacles:

(1) As we noted in Section 3, the CBCMAC-based extractor uses its own previous output as the first block of its next input. Consequently, the CBCMAC inputs are not independent of the seed. This pushes leftover-hash-lemma style results like Theorem 1 out of scope, and furthermore prevents us from employing a black-box hybrid argument to lift the results to the multiple-query setting.

(2) The $\mathcal{O}(L^2)$ term is problematic, contributing a $\mathcal{O}(L/2^{k/2})$ term to bound.² We note that this is significantly worse than the familiar $\mathcal{O}(L^2/2^k)$ “birthday bound”—although the two both become vacuous when $L \approx 2^{k/2}$, the former violates a desired security level $\epsilon \ll 1$ much sooner (hidden constants being equal). The weak bound is exacerbated by the fact that L may grow very quickly in the ISK-RNG during periods of time when the CE buffer is not available.

Analyzing the CBC-MAC Extractor. In this section we present results that allow us to overcome these hurdles, bringing Theorem 1 into scope. In particular, the main technical result of this section is the following theorem. Loosely, it says that we can still obtain a hybrid-like bound, even though a black-box hybrid argument isn’t possible. Moreover, we can avoid the problem of “runaway” input strings (resulting in large L) by, in effect, only counting a fixed-length prefix of such strings.

Theorem 2. *Fix positive integers L , k , q and n with $q \leq n$. For $i \in [1..n]$, let $I_i \in \{0, 1\}^*$ be random variables with lengths divisible by k ,*

² A set of slides published by Intel [17] claims a much stronger result based on Theorem 1. However, in addition to failing to account for point (1) above, the difference appears to stem from a mistake in translating notation. Specifically, the above theorem from [4] writes the second term under the radical as $K \cdot \epsilon(L, K)$, where $\epsilon(L, K) = \mathcal{O}(L^2/K^2)$ and in our notation $K = 2^k$. The Intel slides, however, appear to have mistranscribed this term as $L \cdot \epsilon(L, K)$ (in their notation, $L = b$ and $K = 2^n$). Since $L \ll K$ for values of interest, Intel’s claim significantly underestimates the concrete security bound.

and sample $R_i \xleftarrow{s} \{0,1\}^k$. Fix $\pi \xleftarrow{s} \text{Perm}(k)$. Define I_i^L and I_i^R to be the unique strings such that $|I_i^L| = \min\{|I_i|, Lk\}$ and $I_i = I_i^L I_i^R$. Let $C_i = \text{CBCMAC}_\pi(C_{i-1} \parallel I_i)$, where C_0 is a random variable independent of π and each I_i and R_i . Then $\Delta((\pi, C_1, \dots, C_q, I_{>q}), (\pi, R_1, \dots, R_q, I_{>q})) \leq \frac{1}{2} \sum_{i=1}^q \sqrt{2^{k-\mathbf{H}_\infty(I_i^L \mid I_{>i}, I_i^R)} + \frac{\mathcal{O}((L+1)^2)}{2^k}}$, where $I_{>m} = (I_{m+1}, \dots, I_n)$ for integer m .

The proof is available in the full version of this document [16].

It remains to show that, with high probability, the (potentially) truncated extractor input contains sufficient min-entropy. Note that making reasonable min-entropy assumptions regarding the entropy source is not sufficient; for example, the approximate 1% false-positive rate of the health tests on uniformly random 256-bit strings implies that there are at least 2^{249} *unhealthy* strings. Therefore the entropy source could produce *only* unhealthy samples, resulting in unbounded L , and still have high min-entropy. In order to avoid such pathological behavior, we will later (in Section 7.2) need to introduce additional assumptions regarding the rate at which the entropy source produces healthy samples. Ultimately, we will choose L such that we have a high probability of never needing more than $L/2$ samples, but such that $L/2^{k/2}$ is small, as this term will dominate our security bounds.

5 Modeling the ISK-RNG as a PWI

Building upon DPRVW, here we define the syntax of a PWI. We give the syntax first, and then discuss what it captures, pointing out where our definition differs from DPRVW.

5.1 The PWI Model

Definition 3 (PWI). Let p , and ℓ be non-negative integers, and let IFace , Seed , State be non-empty sets. A PRNG with input (PWI) with interface set IFace , seed space Seed , and state space State is a tuple of deterministic algorithms $\mathcal{G} = (\text{setup}, \text{refresh}, \text{next}, \text{tick})$, where

- setup takes no input, and generates an initial PWI state $S_0 \in \text{State}$. Although setup itself is deterministic, it may be provided oracle access to an entropy source \mathcal{D} , in which case its output S_0 will be a random variable determined by the coins of \mathcal{D} .
- $\text{refresh} : \text{Seed} \times \text{State} \times \{0,1\}^p \rightarrow \text{State}$ takes a seed $\text{seed} \in \text{Seed}$, the current PWI state $S \in \text{State}$, and string $I \in \{0,1\}^p$ as input, and returns new state.
- $\text{next} : \text{Seed} \times \text{IFace} \times \text{State} \rightarrow \text{State} \times (\{0,1\}^\ell \cup \{\perp\})$ takes a seed, the current state, and an interface label $m \in \text{IFace}$, and returns a new state, and either ℓ -bit output value or a distinguished, non-string symbol \perp .
- $\text{tick} : \text{Seed} \times \text{State} \rightarrow \text{State}$ takes a seed and the current state as input, and returns a new state. □

We will typically omit explicit mention of the seed argument to `refresh`, `next` and `tick`, unless it is needed for clarity.

The `setup` algorithm captures the initialization of the PWI, in particular its internal state. Unlike DPRVW, whose syntax requires `setup` to generate the PWI seed, we view the seed as something generated externally and provided to the PWI. Permitting an explicit setup procedure is necessary to correctly model ISK-RNG and, more generally, allows us to formulate an appropriate security definition for PWI initialization.

The `refresh` algorithm captures the incorporation of new entropy into the PWI state. Like DPRVW, we treat the entropy source as external. This provides a clean and general way to model the source as untrusted to provide consistent, high-entropy outputs.

Our `next` algorithm captures the interface exposed to (potentially adversarial) parties that request PWI outputs. By embellishing the DPRVW syntax for `next` with the interface set `interface`, we model APIs that expose multiple functionalities that access PWI state. This is certainly the case for the ISK-RNG, via the `RDRAND` and `RDSEED` instructions, as well as `/dev/[u]random`. We also model blocking by letting `next` return \perp .

The `tick` algorithm is entirely new, and requires some explanation. In the security notions formalized by DPRVW, the passage of “time” is implicitly driven by adversarial queries. (This is typical for security notions, in general.) But real PRNGs like the ISK-RNG may have behaviors that update the state in ways that are not cleanly captured by an execution model that is driven by entropy-input events (`refresh` calls), or output-request events (`next` calls). The `tick` algorithm handles this, while allowing our upcoming security notions to retain the tradition of being driven by adversarial queries: the adversary will be allowed to “query” the `tick` oracle, causing one unit of time to pass and state changes to occur.

5.2 Mapping ISK-RNG into the PWI Model

We now turn our attention to mapping the ISK-RNG specification into the PWI model. Figure 3 summarizes the state that our model tracks. Figure 4 provides our model for the PWI `setup`, `refresh`, `next`, and `tick` oracles. Two additional procedures, `DRBG` and `reseed`, are used internally.

6 PWI Security

Having defined the syntax for PWIs, we can now introduce corresponding security notions. The basic notions are those of DPRVW, with a few notable alterations. To handle issues of non-uniform state and (more) realistic initialization procedures, we introduce a new technical tool, masking functions, that allows us to cleanly address these issues.

Variable	Bits	Description
ESSR	256	Entropy source shift register
window	8	Counts new bits in the ESSR
OSTE ₁	256	
OSTE ₂	256	Online self-tested entropy buffers
CE ₀	128	
CE ₁	128	Conditioned entropy buffers
ptr	1	Tracks CE buffer to condition next
health	256	Tracks health of last 256 ES samples
K	128	DRBG key (For AES-CTR)
IV	128	DRBG IV (For AES-CTR)
out _{1,...,8}	512	Eight 64-bit output buffers
outcount	≥ 4	Counts number of full output buffers
count	≥ 9	Counts DRBG calls since reseeding
CFull	1	Set if CE buffers are available
block	1	Set if reseed has priority over RDSEED

Fig. 3. State variables of the ISK-RNG

6.1 Basic Notions

Here we define four PWI-security notions, in the game-playing framework [2]. In each there is a (potentially adversarial) entropy source \mathcal{D} , and an adversary A . The latter is provided access to the oracles detailed in Figure 5 (top), and what distinguishes the four notions are restrictions applied to the queries of the adversary A . In particular, we consider the following games:

Robustness (ROB): no restrictions on queries.

Forward security (FWD): no queries to set-state are allowed; and a single query to get-state is allowed, and this must be the final query.

Backward security (BWD): no queries to get-state are allowed; a single query to set-state is allowed, and this must be the first query.

Resilience (RES): no queries to get-state or set-state are allowed.

See DPRVW for additional discussion. We note that all games share common initialize and finalize procedures, shown in Figure 5 (bottom). Thus, the robustness-advantage of A in attacking \mathcal{G} is defined to be $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^{\text{rob}}(A) = 2 \Pr[\text{ROB}_{\mathcal{G}, \mathcal{D}}(A) = 1] - 1$. The forward security, backward security, and resilience advantages $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^{\text{fwd}}(A)$, $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^{\text{bwd}}(A)$, and $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^{\text{res}}(A)$ are similarly defined. It is clear that robustness implies forwards and backwards security, and both of these independently imply resilience.

We note that, because the PRNG cannot reasonably be expected to produce random-looking outputs without sufficient entropy or with a known or corrupted state, the various security experiments track (1) a boolean variable `corrupt` and (2) a value γ measuring the total entropy that has been fed into the PRNG since `corrupt` was last set. These serve as book-keeping devices to prevent trivial wins. The `corrupt` flag is cleared whenever γ exceeds some specified threshold γ^* .

Oracle setup(S):

```

01  for  $i = 1, 2, 3, 4$  do
02     $S.\text{CE}_0 \leftarrow \text{CBCMAC}_{K'}(S.\text{CE}_0)$ 
03    while  $S.\text{ptr} = 0$  do
04       $I \leftarrow \text{ES}$ 
05       $S \leftarrow \text{refresh}(S, I)$ 
06     $S.\text{CE}_1 \leftarrow \text{CBCMAC}_{K'}(S.\text{CE}_1)$ 
07    while  $S.\text{ptr} = 1$  do
08       $I \leftarrow \text{ES}$ 
09       $S \leftarrow \text{refresh}(S, I)$ 
10     $S \leftarrow \text{reseed}(S)$ 
11  for  $i = 1, 3, 5, 7$  do
12     $(S, R) \leftarrow \text{DRBG}(S)$ 
13     $S.\text{out}_i \parallel S.\text{out}_{i+1} \leftarrow R$ 
14   $S.\text{outcount} \leftarrow 8$ 
15  return  $S$ 

```

Oracle DRBG(S):

```

16   $S.\text{IV} \leftarrow S.\text{IV} + 1$ 
17   $R \leftarrow \text{CTR}_K^V(0^{128})$ 
18  if  $S.\text{CEfull}$  then
19     $S \leftarrow \text{reseed}(S)$ 
20  else if  $S.\text{count} < 512$ 
21     $S.K \parallel S.V \leftarrow \text{CTR}_{S.K}^{S.V+1}(0^{256})$ 
22     $S.\text{count} \leftarrow S.\text{count} + 1$ 
23  else
24    return  $(S, \perp)$ 
25  return  $(S, R)$ 

```

Oracle tick(S):

```

26  if  $S.\text{CEfull}$  and  $S.\text{count} > 0$  then
27     $S \leftarrow \text{reseed}(S)$ 
28  return  $S$ 
29  if  $S.\text{count} < 512$  then
30    if  $S.\text{outcount} < 8$  then
31       $S.\text{outcount} \leftarrow S.\text{outcount} + 1$ 
32       $(S, R) \leftarrow \text{DRBG}(S)$ 
33       $S.\text{out}_{\text{outcount}} \leftarrow R$ 
34  return  $S$ 
35  return  $S$ 

```

Oracle refresh(S, I):

```

36   $\text{shift}(S.\text{ESSR}, I)$ 
37   $S.\text{window} \leftarrow S.\text{window} + 1 \bmod 256$ 
38  if  $S.\text{window} = 0$  then
39     $\text{shift}(S.\text{health}, \text{isHealthy}(S.\text{ESSR}))$ 
40     $S.\text{OSTE}_2 \leftarrow S.\text{OSTE}_1$ 
41     $S.\text{OSTE}_1 \leftarrow S.\text{ESSR}$ 
42     $i \leftarrow S.\text{ptr}$ 
43     $I_j^i \leftarrow I_j^i \parallel S.\text{OSTE}_2$  // Record-keeping
44     $S.\text{CE}_i \leftarrow \text{CBCMAC}_{K'}^{S.\text{CE}_i}(S.\text{OSTE}_2)$ 
45  if  $\text{sum}(S.\text{health}) \geq 128$  then
46    if  $\text{isHealthy}(S.\text{OSTE}_2)$  then
47       $S.\text{samples} \leftarrow S.\text{samples} + 1$ 
48  if  $S.\text{samples} = m$  then
49     $S.\text{samples} \leftarrow 0$ 
50  if  $S.\text{ptr} = 0$  then
51     $S.\text{ptr} \leftarrow 1$ 
52  else
53     $S.\text{ptr} \leftarrow 0; S.\text{CEfull} \leftarrow 1$ 
54     $C_j^0 \parallel C_j^1 \leftarrow S.\text{CE}$  // Record-keeping
55     $j \leftarrow j + 1$ ; // Record-keeping
56  return  $S$ 

```

Oracle reseed(S):

```

57   $S.K \parallel S.V \leftarrow \text{CTR}_K^{V+1}(S.\text{CE})$ 
58   $S.\text{CE}_0 \leftarrow \text{CBCMAC}_{K'}(S.\text{CE}_0)$ 
59   $S.\text{CE}_1 \leftarrow \text{CBCMAC}_{K'}(S.\text{CE}_1)$ 
60   $S.\text{count} \leftarrow 0; S.\text{CEfull} \leftarrow 0$ 
61   $S.\text{ptr} \leftarrow 0; S.\text{block} \leftarrow 0$ 
62  return  $S$ 

```

Oracle next(interface, S):

```

63  if interface = RDRAND then
64    if  $S.\text{outcount} = 0$  then return  $(S, \perp)$ 
65     $R \leftarrow \text{LSB}_{64}(S.\text{out}_1)$ 
66    for  $i = 1, \dots, 7$  do
67       $S.\text{out}_i \leftarrow S.\text{out}_{i+1}$ 
68     $S.\text{outcount} \leftarrow S.\text{outcount} - 1$ 
69  return  $(S, R)$ 
70  else if interface = RDSEED
71    if  $S.\text{CEfull} = 0$  then
72      return  $(S, \perp)$ 
73    if  $S.\text{block} = 1, S.\text{count} > 0$  then
74      return  $(S, \perp)$ 
75     $R \leftarrow S.\text{CE}_0 \parallel S.\text{CE}_1$ 
76     $S.\text{CEfull} \leftarrow 0; S.\text{ptr} \leftarrow 0$ 
77     $S.\text{CE}_0 \leftarrow \text{CBCMAC}_{K'}(S.\text{CE}_0)$ 
78     $S.\text{CE}_1 \leftarrow \text{CBCMAC}_{K'}(S.\text{CE}_1)$ 
79     $S.\text{block} \leftarrow 1$ 
80  return  $(S, R)$ 

```

Fig. 4. The above oracles describe the behavior of ISK-RNG from within the PWI model. See Table 3 for a description of the state variables $S.*$. All bits are initially zero. For Ivy Bridge chips, $m = 2$, and for Broadwell chips $m = 3$. The key $K' = \text{AES}_0(1)$ is fixed across all chips. The function $\text{shift}(x, y)$ sets value of x to the right-most $|x|$ bits of $x \parallel y$. Lines marked with a “Record-keeping” comment are there to aid in proofs and exposition.

Oracle \mathcal{D}-refresh:	Oracle $\text{next-ror}(m)$:	Oracle $\text{get-next}(m)$:	Oracle get-state:
$(\sigma, I, \gamma, z) \xleftarrow{\$} \mathcal{D}(\sigma)$ $S \leftarrow \text{refresh}(S, I)$ $c \leftarrow c + \gamma$ if $c \geq \gamma^*$ then corrupt \leftarrow false return (γ, z)	if corrupt then return \perp $(S, R_0) \leftarrow \text{next}(m, S)$ if $R_0 = \perp$ then $R_1 \leftarrow \perp$ else $R_1 \xleftarrow{\$} \{0, 1\}^\ell$ return R_b	$(S, R) \leftarrow \text{next}(m, S)$ if corrupt then $c \leftarrow 0$ return R	$c \leftarrow 0$ corrupt \leftarrow true return S
Procedure initialize:	Oracle ES:	Oracle wait:	Oracle set-state(S^*):
$\sigma \leftarrow 0$; seed $\xleftarrow{\$}$ Seed; $i \leftarrow 0$ $S \leftarrow \text{setup}^{\text{ES}}$ $c \leftarrow n$; corrupt \leftarrow false $b \xleftarrow{\$} \{0, 1\}$ return (seed, $(\gamma_j, z_j)_{j=1}^i$)	$i \leftarrow i + 1$ $(\sigma, I, \gamma_i, z_i) \xleftarrow{\$} \mathcal{D}(\sigma)$ return I	$S \leftarrow \text{tick}(S)$ return ε	$c \leftarrow 0$ corrupt \leftarrow true $S \leftarrow S^*$
Procedure finalize(b):			
			if $b = b^*$ then return 1 else return 0

Fig. 5. **Top:** Oracles for the PWI security games. **Bottom:** the shared initialize and finalize procedures for the PWI security games. Recall that the output of initialize is provided to adversary A as input, and the output of finalize is the output of the game.

6.2 Masking Functions and Updated Security Notions

As noted earlier, the DPRVW security definitions assume the PWI state is initially uniformly random. However, this does not realistically model the behavior of real-world PWIs, notably ISK-RNG, which do not attempt to reach a pseudorandom state; for example, they may maintain counters. (Indeed one can construct PWIs that would be deemed secure when starting from a uniformly random state, but that would not be secure in actuality; the reverse is also true. See the full version of this paper [16] for examples.) Yet, clearly, some portion of the PWI state must be unpredictable to an attacker, as otherwise one cannot expect PWI outputs to look random.

To better capture real-world characteristics of PWI state, we introduce the idea of a *masking function*. A masking function M over state space State is a randomized algorithm from State to itself. As an example, if states consist of a counter c , a fixed identifier id, and a buffer B of (supposedly) entropic bits, then $M(c, \text{id}, B)$ might be defined to return (c, id, B') where B' is sampled by M from some distribution.

A masked state is meant to capture whatever characterizes a “good” state of a PWI, i.e. after it has accumulated a sufficient amount of externally provided entropy. Informally, for any state S , we want that (1) a PWI with state $M(S)$ should produce pseudorandom outputs, and (2) after the PWI has gathered sufficient entropy, its state S should be indistinguishable from $M(S)$.

To the second point, the initial PWI state S_0 is of particular importance. In the following definition, we characterize masking functions M such that the initial S_0 and $M(S_0)$ are indistinguishable.

Definition 4 (Honest-initialization masking functions). Let \mathcal{D} be an entropy source, $\mathcal{G} = (\text{setup}, \text{refresh}, \text{next})$ be a PWI with state space State, A be an adversary, and $M : \text{State} \rightarrow \text{State}$ be a masking function. Let (seed, Z) be the random variable returned by running the `initialize()` (Figure 5) using \mathcal{G} and \mathcal{D} , and let S_0 be the state produced by this procedure. Set $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}, M}^{\text{init}}(A) = \Pr[A(S_0, \text{seed}, Z) \Rightarrow 1] - \Pr[A(M(S_0), \text{seed}, Z) \Rightarrow 1]$. If $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}, M}^{\text{init}}(A) \leq \epsilon$ for any adversary A running in time t , then M is a $(\mathcal{G}, \mathcal{D}, t, \epsilon)$ -honest-initialization masking function. \square

Note that the above definition is made with respect to a specific \mathcal{D} . The assumptions required of \mathcal{D} (e.g., that it will provide a certain amount of entropy within a specified number of queries) will depend on the PWI in question, but should be as weak as possible.

We now define “bootstrapped” versions of the PWI security goals, which always begin from a masked state. This will allow us to reason about security when the PWI starts from an “ideal” state, i.e. what we expect after a secure initialization of the system.

Definition 5 (Bootstrapped security). Let \mathcal{G} be a PWI and M be a masking function. For $x \in \{\text{fwd}, \text{bwd}, \text{res}, \text{rob}\}$, let $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^{x/M}(A)$ be defined as $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^x(A)$, except with line 02 of the `initialize` procedure (Fig. 5) changed, to execute instead $S' \xleftarrow{s} \text{setup}^{\text{ES}}; S \xleftarrow{s} M(S')$. \square

6.3 PWI-Security Theorems

Bootstrapped security notions are useful, because they allow the analysis to begin with an idealized state. However, this comes at a cost: we need to ensure that the masking function is honest in the sense that it accurately reflects the result of running the `setup` procedure. The following theorem states the intuitive result that if the masking function is secure (and honest), then security when the PWI begins in a masked state $M(S)$ implies security when the PWI begins in state S . We omit the simple proof, which follows from a standard reduction argument.

Theorem 3. Let \mathcal{G} be a PWI, \mathcal{D} be an entropy source, and M be a masking function. Suppose M is a $(\mathcal{G}, \mathcal{D}, t, \epsilon)$ -honest initialization mask. Then for any $x \in \{\text{fwd}, \text{bwd}, \text{res}, \text{rob}\}$ there exists some adversary $B(\cdot)$ such that for any adversary A , $\mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^x(A) \leq \mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^{x/M}(B(A)) + \epsilon$. Further, if it takes time t' to compute M , and A makes q queries and runs in time t , then $B(A)$ makes q queries and runs in time $\mathcal{O}(t) + t'$.

For a second general result, we revisit a nice theorem by DPRVW and adapt it to our model. The theorem states that if a PWI possesses two weaker security properties—roughly, the ability to randomize a corrupted state after harvesting sufficient entropy and the ability to keep its state pseudorandom in the presence of adversarial entropy—then it is robust. These definitions, however, again assume that a state “should” appear uniformly random. We present modified definitions

that instead use masking functions, and prove an analogous theorem. While the transition involves a couple subtleties—in particular, we require an idempotence property of the masking function—the proof is essentially identical to the one in [5]; therefore we make an informal statement here and defer the formal treatment to the full version [16].

Theorem 4 (Informal). *Let \mathcal{G} be a PWI. Suppose there exists a mask M such that: (1) When starting from an arbitrary initial state S of the adversary’s choosing, the final PWI state S' is indistinguishable from $M(S')$ provided the PWI obtains sufficient entropy; (2) When starting from an initial state $M(S)$ (for adversarially chosen S), the final PWI state S' is indistinguishable from $M(S')$, even if the adversary controls the intervening entropy input strings; (3) \mathcal{G} produces pseudorandom outputs when in a masked state. Then \mathcal{G} is robust.*

7 Security of the ISK-RNG as a PWI

We are now positioned to analyze the security of ISK-RNG. To begin, we demonstrate some simple attacks that violate both forwards and backwards security (hence robustness, too). Next, we show that by placing a few additional restrictions on adversaries—restrictions that are well-motivated by the hardware—we can recover forward security. As we said in our introduction, the concrete security bounds we prove are not as strong as one might hope, due to some limitations of CBCMAC’s effectiveness as an entropy extractor in the ISK-RNG. However, we are able to prove somewhat better results when legitimate parties use only the `RDRAND` interface, even when attackers also have access to `RDSEED`. This means that, e.g., a hostile process can’t use its access to `RDSEED` to learn information about `RDRAND` return values used by a would-be victim; the result also implies stronger results for Ivy Bridge chips, where `RDSEED` is not available.

For the remainder of Section 7, we fix the following constants: $p = 1$ is the length of each entropy input; $k = 128$ is the length of each CBCMAC input block (since ISK-RNG uses AES); $\text{IFace} = \{\text{RDSEED}, \text{RDRAND}\}$ are the ISK-RNG interfaces; $m = 2, 3$ is the number of healthy samples required by Ivy Bridge and Broadwell, respectively, before the CE buffer is available; and $\ell = 64$ is the length of the PWI outputs. Although `RDRAND` also allows programs to request 16 or 32 bits, this is implemented by fetching then truncating a 64-bit output, and similarly with `RDSEED` [12]. Therefore we assume without loss of generality that the adversary only requests the full 64 bits.

Recall that in the PWI model, the entropy source leaks information γ about each input string. We assume that every 256th such string (each one a single bit, $p = 1$) leaks the health of the corresponding 256 bit string (as determined by the online health test). Hence the adversary will always know the health of the OSTE buffers and the value of the `health` buffer. This is not simply a convenience: because the CE buffer is not available until it has been reconditioned with m healthy samples, `RDSEED` may leak health information through a timing side channel.

When the CE buffer is available, it can be used to reseed the DRBG or to service a `RDSEED` instruction. Priority is given to whichever was not last used [12].

However, because the PWI model cannot describe pending RDSEED instructions, the adversary must explicitly use its `wait` oracle to yield when it has priority: a `wait` invocation uses the CE to reseed, while a RDSEED invocation returns its contents.

The adversary’s `wait` oracle also allows us to account for the fact that updating the eight 64-bit output buffers is not an atomic operation. By using the tick function (invoked by `wait`) to only fill one at a time, we conservatively allow the adversary to control if a reseeding operation intervenes. Note that `tick` will reseed rather than fill an output buffer if reseeding is desired ($S.\text{count} > 0$) and possible ($S.\text{CEfull} = 1$). This reflects the priorities of the hardware [12].

In order to save power, the entropy source goes to sleep if all the output buffers are full, the CE buffer is available, and no RDRAND instructions have been processed since the last reseed [12]. The PWI model, however, requires that we continue to provide \mathcal{D} -refresh access to the adversary. Our decision to leak health information to the adversary allows us to avoid any problems here: the adversary knows when the entropy source sleeps, so we can restrict the adversary to not make \mathcal{D} -refresh calls when it does.

To make this power-saving hardware constraint “work” with the PWI model, we assume that each healthy 256-bit block produced by the entropy source contains at least γ bits of min-entropy. Formally, define $(\sigma_i, b_i, \gamma_i, z_i) = \mathcal{D}(\sigma_{i-1})$ for $i \geq 1$ (where $\sigma_0 = \varepsilon$), and let $I_i = b_{256i}b_{256i+1}\dots b_{256i+255}$. We assume $\mathbf{H}_\infty(I_i \mid (\sigma_j, I_j, \gamma_j, z_j)_{j \neq i}, I_i \in \mathcal{H}) \geq \gamma$, for some $\gamma > 0$, and require that $\sum_{j=256i+255}^{256i+255} \gamma_j \geq \gamma$ whenever $I_i \in \mathcal{H}$. We set $\gamma^* = m\gamma$ to demand, in effect, that ISK-RNG delivers on its implicit promise that m healthy entropy samples are sufficient. At the end of this section, we will draw from the CRI report’s analysis to find reasonable estimates for γ and discuss the implications.

7.1 Negative Results

We begin with some quick negative results, showing that the ISK-RNG achieves neither forward nor backwards security. This immediately rules out robustness, too. We again emphasize that these negative results will be followed by positive results for realistic classes of restricted adversaries; we present them primarily to motivate the coming restrictions.

Theorem 5 (ISK-RNG lacks forward security). *There exists an adversary A making one next-ror query and one get-state query such that for any entropy source \mathcal{D} , $\mathbf{Adv}_{\text{ISK}, \mathcal{D}}^{\text{fwd}}(A) = 1 - 2^{-128}$.*

Theorem 6 (ISK-RNG lacks backward security). *There exists an adversary A making one next-ror query and one set-state query such that for any entropy source \mathcal{D} , $\mathbf{Adv}_{\text{ISK}, \mathcal{D}}^{\text{bwd}}(A) = 1 - 2^{-128}$.*

In the case of backwards security, the adversary sets some initial state S with $S.\text{samples} = 0$, makes a sequence of \mathcal{D} -refresh calls to clear the `corrupt` flag (which, by our previously stated assumptions, will happen as soon as the CE buffer becomes available), and finally assigns $X \leftarrow \text{next-ror(RDRAND)}$. The adversary then checks if

$X = S.\text{out}_1$, and outputs 0 if this is the case and 1 otherwise. For forward security, the adversary assigns $X \leftarrow \text{next-ror}(\text{RDSEED})$, then learns the resulting state S using $\text{get-state}()$. If $X = \text{AES}_0^{-1}(S.\text{CE}_0) \parallel \text{AES}_0^{-1}(S.\text{CE}_1)$, the adversary outputs 0; otherwise, the it outputs 1. (Here, $\mathbf{0} = 0^{128}$.)

However, these results are very conservative. In the case of forward security, the hardware will quickly recondition the CE buffer and refill the output buffers, effectively erasing all state that could be used to compute previous outputs. Backwards security is more complicated because not only do future outputs persist in the output buffer indefinitely, but future DRBG keys are leaked via the ESSR, OSTE, and CE buffers. Once the output buffers are flushed, though, these other buffers will quickly be overwritten with fresh entropy.

7.2 Positive Results

We now turn our attention to restricted, but still conservative, classes of adversary in order to produce positive results.

Additional assumptions. We further assume that in the forward-security game, adversaries do not make their get-state query until they have allowed the output buffers to be refilled. This assumption is motivated by the speed with which the hardware will automatically accomplish this: at the reported `RDRAND` throughput of 500 MB/s, all eight 64-bit buffers can be refilled around 8 million times per second. Formally:

Definition 6 (Delayed adversaries). *An adversary A attacking ISK-RNG in the forward-security game is delayed if after making its last `get-next` and `next-ror` queries, A calls $\mathcal{D}\text{-refresh}$ until the CE buffer is available, then calls `wait` nine times before making its `get-state` query.* \square

This will trigger a reseed and then refill any empty output buffers.

Moreover, we will assume there is some positive probability β such that each 256-bit block of bits from the entropy source is healthy with probability at least β . Formally (recall that $\mathcal{H} \subseteq \{0, 1\}^{256}$ is the set of strings deemed healthy by ISK-RNG's online health tests):

Definition 7 (β -healthy). *Let \mathcal{D} be an entropy source and fix $\beta > 0$. Let $\mathcal{H} \subseteq \{0, 1\}^{256}$ be the set of strings deemed healthy by the ISK-RNG. For $i = 1, 2, 3, \dots$ define $(\sigma_i, b_i, \gamma_i, z_i) = \mathcal{D}(\sigma_{i-1})$ (where $\sigma_0 = \varepsilon$), and for $j = 0, 1, 2, \dots$, define $B_j = b_{256j} \parallel b_{256j+1} \parallel \dots \parallel b_{256j+255}$. Let $H_j = 1$ if $B_j \in \mathcal{H}$, and set $H_j = 0$, otherwise. Then \mathcal{D} is β -healthy if for all such j and all $H \in \{0, 1\}^{j-1}$, $\Pr[B_j \in \mathcal{H} | (H_\ell)_{\ell < j} = H] \geq \beta$.* \square

So for any positive integers ℓ and L_m , we can upper bound the probability that the sequence $(B_i)_{i=\ell}^{\ell+L_m-1}$ contains fewer than m healthy values using: $\Pr[|\{j : B_j \in \mathcal{H}, \ell \leq j < \ell + L_m\}| < m] \leq \sum_{i=0}^{m-1} \binom{L_m}{i} \beta^i (1 - \beta)^{L_m - i}$.

k	CBCMAC blocksize (128 bits)
m	Number of “healthy” $2k$ -bit strings that need to be conditioned before the CE buffer becomes available ($m = 2, 3$ for Ivy Bridge and Broadwell chips, respectively).
L_m	Parameter we can freely choose to keep both $\hat{\epsilon}(L_m)$ and $\epsilon(L_m)$ small.
γ	An assumed lower bound on the conditional min-entropy of healthy strings.

Fig. 6. Summary of values used for theorem statements

Remark 1. Our goal is to identify under what reasonable assumptions ISK-RNG could be deemed secure, and, as we argued at the end of Section 4, this requires making an assumption about the entropy source’s ability to produce “healthy” samples (a min-entropy assumption is too weak). We settled on the above β -healthy assumption because it is simple and fairly broad: we do not assume the probabilities of samples being healthy are constant or even independent, just that the conditional probabilities don’t dip below the β threshold. Moreover, we later show that the “unhealthy sample rate” could easily be fifty times the ideal 1% false-positive base rate without significantly damaging our bounds. Finally, even the β -healthy assumption is more than we need. We require an upperbound on the probability on the left-hand side of the above equation, and the β -healthy assumption provides a natural, concrete way to think about this probability.

Rigorously testing the β -healthy assumption without access to the entropy source is problematic. That being said, barring such access, we doubt it would be possible to do significantly better.

With these assumptions, we are ready to continue on to our positive results. Our first step is to define an appropriate masking function that describes an “ideal” state, and then to prove that `setup` creates such a state. This lets later proofs simply assume we begin in an idealized state (see Theorem 3).

ISK-RNG masking function. Fix the masking function $M : \{0, 1\}^n \rightarrow \{0, 1\}^n$ that on input S , overwrites $S.\text{CE}$, $S.K$, $S.\text{IV}$, and $S.\text{out}_{1,\dots,8}$ with independent, uniformly random strings of the appropriate lengths, leaves all other portions of the state untouched, and returns the result (refer back to Fig. 3 for a listing of the components of the ISK-RNG state S). This is the ISK-RNG masking function.

Recall the results of Theorem 2. For convenience, we define $\epsilon(L_m) = \mathcal{O}(L_m + 1)/2^{k/2}$ and $\hat{\epsilon}(L_m) = \sum_{i=0}^{m-1} \binom{L_m}{i} \beta^i (1 - \beta)^{L_m - i}$, where $\epsilon(L_m)$ is from Theorem 2 and $\hat{\epsilon}(L_m)$ is the above bound on the probability of obtaining fewer than m healthy samples from a β -healthy entropy source within L_m trials. Our theorem statements refer to various previously defined values, summarized in Figure 6.

The following lemma says that if AES is a secure PRP (against adversaries making three queries) and each healthy sample from the entropy source has sufficiently large min-entropy, then the ISK-RNG masking function is honest. That is, that the ISK-RNG setup procedure successfully places the hardware in a state where (we will show) it can begin producing pseudorandom outputs.

Lemma 1 (ISK-RNG masking function is honest). Fix positive integers k and m , and fix $0 < \beta \leq 1$. Let L_m be a positive integer. Let M be the ISK-RNG masking function. Let \mathcal{D} be a β -healthy entropy source. Then for any adversary A , there exists an adversary B running in the same time and making three queries such that $\mathbf{Adv}_{\text{ISK}, \mathcal{D}, M}^{\text{init}}(A) \leq 2^{(k-m\gamma)/2+2} + 4\epsilon(L_m) + 8\epsilon(L_m) + 5(\mathbf{Adv}_{\text{AES}}^{\text{PRP}}(B) + \frac{3}{2^k})$.

The proof is deferred to the full version of this paper [16]. Using reasonable estimates for the big- \mathcal{O} constant and γ (discussed in Section 7.3) provides us with an upper bound of roughly 2^{-60} for the first three terms of the security bound for both $m = 2, 3$.

Remark 2. The PRP term may be problematic if one takes the view that RDSEED should offer information-theoretic security. That is, Lemma 2 says that the ISK-RNG initialization procedure yields state—which includes the CE buffers—that is only computationally indistinguishable from “ideal”. However, we observe that if one adjusts the masking function to leave the output buffers unchanged, and demands a post-setup reconditioning (which the hardware endeavors to provide, anyway), one could indeed use the result to prove information-theoretic RDSEED security. However, this would be at the expense of *not* being able to prove security of the RDRAND interface, a task which necessarily requires computational assumptions.

Forward Security. Our exploration of forward security proceeds in two steps. To begin, we introduce a new game, $M\text{-RDRAND}$, which differs from $M\text{-FWD}$ in that the `next-ror` oracle always returns the “real” value R_0 when queried on the RDSEED interface, but behaves normally during queries to the RDRAND interface. Define

$$\mathbf{Adv}_{\mathcal{G}, \mathcal{D}}^{\text{fwd-RDRAND}/M}(A) = 2 \Pr[M\text{-RDRAND}(A) \Rightarrow 1] - 1.$$

Proving the security of this game is not only a useful intermediate step in proving the security of $M\text{-FWD}$, but also can be interpreted as measuring the strength of RDRAND return values when an adversary also has access to the RDSEED instruction (which can be used to learn information about the ISK-RNG state, but that we do not require to return pseudorandom values). This distinction is valuable, because the concrete bounds on the $M\text{-FWD}$ experiment are not as strong as one would hope.

Theorem 7 ($M\text{-RDRAND}$). Let A be a delayed adversary making q queries to RDRAND and running in time t . Then there exists an adversary B making three queries and running in time $\mathcal{O}(t)$ such that $\mathbf{Adv}_{\text{ISK}, \mathcal{D}}^{\text{fwd-RDRAND}/M}(A) \leq 2(q + 4)(\mathbf{Adv}_{\text{AES}}^{\text{PRP}}(B) + \frac{3}{2^k})$.

The proof appears in the full version [16]. Barring an efficient attack on AES (that only uses three queries!) this bound is quite strong. If q were to grow quite large, say on the order of $q \approx 2^{80}$, then the bound might begin to approach 2^{-40} , which seems a reasonable safety margin. However, even at the reported rate of around 500 MB/s, ISK-RNG would take over 70 years to reach this point. Moreover, the hybrid factor of q is likely a conservative artifact of the proof.

Note, however, that this bound applies to ISK-RNG when starting in an “ideal” masked state; one needs to add in the bound from Lemma 1 to account for initialization. As we mentioned earlier, reasonable estimates for the big- \mathcal{O} constant and γ (see Section 7.3) place this term at roughly 2^{-60} .

We now proceed to the “full” forward-security result, where both the RDRAND and the RDSEED interfaces are required to produce indistinguishable-from-random outputs. Since RDSEED reads directly from the CE buffer, this bound relies more heavily on the entropy source and CBCMAC extractor (and less on the computational security of AES). Again, see the full version [16] for a proof.

Theorem 8 (ISK-RNG’s masked forward security). *Fix a positive integers k and m , and fix $0 < \beta \leq 1$. Let L_m be a positive integer. Let A be a delayed adversary making a combined q queries to get-next and next-ror. Then if \mathcal{D} is β -healthy, there exists some adversary B making three queries and running in the same time as A such that*

$$\begin{aligned} \mathbf{Adv}_{\text{ISK}, \mathcal{D}}^{\text{fwd}/M}(A) &\leq (q+1) \left(2^{(k-m\gamma)/2} + \epsilon(L_m) + 2\hat{\epsilon}(L_m) \right) \\ &\quad + 2(q+4) \left(\mathbf{Adv}_{\text{AES}}^{\text{prp}}(B) + \frac{3}{2^k} \right). \end{aligned}$$

Corollary 1. *Let A be a delayed adversary making a combined q queries to its get-next and next-ror oracles. If \mathcal{D} is β -healthy, then there exists an adversary B making three queries and running in the same time as A such that*

$$\begin{aligned} \mathbf{Adv}_{\text{ISK}, \mathcal{D}}^{\text{fwd}}(A) &\leq (q+5) \left(2^{(k-m\gamma)/2} + \epsilon(L_m) + 2\hat{\epsilon}(L_m) \right) \\ &\quad + (2q+13) \left(\mathbf{Adv}_{\text{AES}}^{\text{prp}}(B) + \frac{3}{2^k} \right), \end{aligned}$$

where the remaining quantities are defined as in Theorem 8.

The corollary follows from applying Theorem 3 to Theorem 8 and Lemma 1. We defer our discussion of this bound to Section 7.3. First, we briefly turn our attention to the questions of backwards security and robustness.

Backwards security and Robustness. The issue with obtaining backwards security (and hence robustness) is that future outputs can linger in the output buffers indefinitely: the hardware will shutdown the entropy source after all the buffers are full and the CE buffer is available. Hence, state remains compromised until fresh entropy filters through the ESSR \rightarrow OSTE₁ \rightarrow OSTE₂ \rightarrow CE buffers and is used to reseed the DRBG, without first being siphoned off by RDSEED.

Consider the worst-case scenario for Ivy Bridge chips, where only the RDRAND interface is available. Following a state compromise, the next eight outputs are revealed by the output buffers, the next 511 may be computed using the compromised DRBG seed, the next 511 may be computed using a DRBG seed determined by the compromised CE buffer, and the next 511 may be computed using a DRBG

key determined by the compromised OSTE and ESSR buffers. This amounts to slightly more than 12 KB of outputs that an adversary could potentially predict.

However, we show in the full version [16] that if one restricts the model to “read-only” adversaries (by denying adversaries access to `set-state` but permitting access to `get-state`) *and* one discounts wins based on the above attacks (by denying adversaries access to `next-ror` until after the “corrupted” values have already been replaced) then ISK-RNG is secure. The concrete bounds we obtain are essentially identical to those provided by Theorems 7 and 8, depending on whether or not one requires the `RDSEED` interface to be secure. See the appendix for further discussion of how these restrictions can be interpreted along with a formal theorem statement and proof.

7.3 Discussion of Results

Let us examine the bound of Corollary 1 in detail. We specialize to the parameters used by Intel: $k = 128$ (a consequence of using AES), $m = 2$ for Ivy Bridge chips, and $m = 3$ for Broadwell chips.

To estimate γ , we turn to the CRI report [8]. Hamburg, Kocher, and Marson subjected raw entropy source bits (using data provided by Intel) to a battery of statistical tests. Using a Markov model with 12 bits of state, they estimate the entropy source produces approximately 0.65 bits of min-entropy per bit of output. However, this was an average (some states of the Markov model resulted in more predictable bits), and a 12-bit state, though perhaps necessary to collect enough samples for a meaningful empirical analysis, is not enough for our purposes. Therefore let us suppose a more conservative rate of 0.5, leading to $\gamma = 128$.

This sets the $(q+5)2^{(k-m\gamma)/2}$ term of our bound to $(q+5)2^{-64}$ for Ivy Bridge (where $m = 2$) and $(q+5)2^{-128}$ for Broadwell (where $m = 3$). The latter bound is quite strong, but, given how quickly q can grow, the former may be worrisome if one wishes to maintain strong security guarantees (e.g., one wishes to cap an adversary’s advantage at 2^{-40}). But this is not the dominate term.

We next consider the term $(q+5)(\epsilon(L_m) + 2\hat{\epsilon}(L_m))$. If we set the big-O constant of ϵ to c (so $\epsilon(L_m) = cL/2^{64}$) then we can choose L_m to optimize this expression. Taking $\beta = 1/2$, $c = \sqrt{10}$, which we believe to be conservative,³ gives an upper bound of $(q+5)2^{-56}$; a more generous $\beta = 0.99$, $c = 1$ improves the upper bound to about $(q+5)2^{-60}$. (These bounds are accurate for both $m = 2$ and $m = 3$, although the corresponding values for L_m differ considerably.)

At this point, limiting an adversary’s advantage to 2^{-40} is difficult — an adversarial process gathering random bits at the benchmarked rate of 500 MB/s could issue the maximum allowable number of queries in under one millisecond. Or at least, this is the case if we demand that `RDSEED` produces uniform random outputs. On the other hand, if one only needs `RDRAND` to be secure, then Theorem 7 suggests that limiting an adversary’s advantage to 2^{-40} is entirely reasonable; in

³ An author of [4] assures us that the asymptotic constant is “certainly less than 10” (and our c is the square root of this constant). A perfect entropy source would give $\beta = 0.99$ since the health tests have a 0.01 false-positive rate.

this setting, we only pick up a single $4(\epsilon(L_m) + 2\hat{\epsilon}(L_m))$ term even after moving to the unmasked forward-security setting, with no troublesome multiplicative factor of q .

The remaining term, $(2q + 13)(\mathbf{Adv}_{\text{AES}}^{\text{prp}}(B) + 3/2^{128})$, is likely to be negligible (recall that B is permitted only three queries).

Our analysis does not point to any obvious, practical attacks (aside from the trivial ones that exploit the output buffers, though it seems a stretch to deem those practical). However, it exposes the CBCMAC extraction process as the likely weakest link, and quantifies the extent of that weakness. An actual attack would need to exploit how the specific output distribution of the entropy source interacts with CBCMAC under the fixed key K' .

7.4 Discussion of the Attack Model

The DPRVW syntax and security notions, which we take as our starting point, assume a strongly adversarial operating environment. They treat the entropy source as adversarial (although not pathologically bad), and allow attackers to observe, even corrupt, the full internal state of the PWI. One might argue that these choices are inappropriate in the case of ISK-RNG. After all, the entire RNG is implemented in 22-32nm hardware, so direct observation of the internal state should require the use of expensive and highly technical equipment, e.g. a state of the art scanning/tunnelling electron microscope.

We are sympathetic to this argument, but still find value in adopting the strong attack model. Even if the entropy source is beyond attacker influence, treating it as adversarial can be seen as a mathematical tool for minimizing the assumptions we make regarding its behavior. Moreover, the model allows us to explore the limits of ISK-RNG’s security, providing analysis of less pessimistic settings (i.e. resilience security) as a byproduct.

Acknowledgments. The authors wish to thank DJ Johnston and Jesse Walker of Intel for answering our questions regarding the design and implementation details of ISK-RNG. Both Terashima and Shrimpton were supported by NSF grants CNS-0845610 and CNS-1319061.

References

1. Barak, B., Halevi, S.: A model and architecture for pseudo-random generation with applications to /dev/random. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, pp. 203–212. ACM (2005)
2. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
3. Chevassut, O., Fouque, P.-A., Gaudry, P., Pointcheval, D.: The Twist-AUGmented Technique for Key Exchange. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 410–426. Springer, Heidelberg (2006)

4. Dodis, Y., Gennaro, R., Håstad, J., Krawczyk, H., Rabin, T.: Randomness Extractions and Key Derivation Using the CBC, Cascade and HMAC Modes. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 494–510. Springer, Heidelberg (2004)
5. Dodis, Y., Pointcheval, D., Ruhault, S., Vergnaud, D., Wichs, D.: Security analysis of pseudo-random number generators with input: /dev/random is not robust. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 647–658. ACM (2013)
6. Everspaugh, A., Zhai, Y., Jellinek, R., Ristenpart, T., Swift, M.: Not-so-random numbers in virtualized Linux and the Whirlwind RNG. In: IEEE Symposium on Security And Privacy (2014)
7. Guterman, Z., Pinkas, B., Reinman, T.: Analysis of the Linux random number generator. In: 2006 IEEE Symposium on Security and Privacy, p. 15. IEEE (2006)
8. Hamburg, M., Kocher, P., Marson, M.E.: Analysis of Intel's Ivy Bridge digital random number generator (2012). http://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.pdf
9. Heninger, N., Durumeric, Z., Wustrow, E., Alex Halderman, J.: Mining your ps and qs: Detection of widespread weak keys in network devices. In: USENIX Security Symposium, pp. 205–220 (2012)
10. Hofemeier, G.: Intel Digital Random Number Generator (DRNG) software implementation guide (August 2012). <https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide> (accessed May 2014)
11. Hofemeier, G., Chesebrough, R.: Introduction to Intel AES-NI and Intel Secure Key instructions (July 2012). <https://software.intel.com/en-us/articles/introduction-to-intel-aes-ni-and-intel-secure-key-instructions> (accessed May 2014)
12. JD Johnston (Intel). Personal communication (May 2014)
13. Lacharme, P., Röck, A., Strubel, V., Videau, M.: The Linux pseudorandom number generator revisited. IACR Cryptology ePrint Archive **2012**, 251 (2012)
14. Mechalas, J.: The difference between RDRAND and RDSEED (November 2012). <https://software.intel.com/en-us/blogs/2012/11/17/the-difference-between-rdrand-and-rdseed> (accessed April 2014)
15. Radhakrishnan, J., Ta-Shma, A.: Bounds for dispersers, extractors, and depth-two superconcentrators. SIAM Journal on Discrete Mathematics **13**(1), 2–24 (2000)
16. Shrimpton, T., Seth Terashima, R.: A provable security analysis of Intel's Secure Key RNG. Cryptology ePrint Archive, Report 2014/504 (2014). <http://eprint.iacr.org/>
17. Walker, J.: Conceptual foundations of the Ivy Bridge random number generator. http://www.ists.dartmouth.edu/docs/walker_ivy-bridge.pdf (November 2011)

A Formal Treatment of Backdoored Pseudorandom Generators

Yevgeniy Dodis^{1(✉)}, Chaya Ganesh¹, Alexander Golovnev¹,
Ari Juels², and Thomas Ristenpart³

¹ Department of Computer Science, New York University, New York, USA
`dodis@cs.nyu.edu, {chaya.ganesh,alexgolovnev}@gmail.com`

² Jacobs Institute, Cornell Tech, New York, USA
`juels@cornell.edu`

³ Department of Computer Sciences, University of Wisconsin, Madison, USA
`tomrist@gmail.com`

Abstract. We provide a formal treatment of backdoored pseudorandom generators (PRGs). Here a saboteur chooses a PRG instance for which she knows a trapdoor that allows prediction of future (and possibly past) generator outputs. This topic was formally studied by Vazirani and Vazirani, but only in a limited form and not in the context of subverting cryptographic protocols. The latter has become increasingly important due to revelations about NIST’s backdoored Dual EC PRG and new results about its practical exploitability using a trapdoor.

We show that backdoored PRGs are equivalent to public-key encryption schemes with pseudorandom ciphertexts. We use this equivalence to build backdoored PRGs that avoid a well known drawback of the Dual EC PRG, namely biases in outputs that an attacker can exploit without the trapdoor. Our results also yield a number of new constructions and an explanatory framework for why there are no reported observations in the wild of backdoored PRGs using only symmetric primitives.

We also investigate folklore suggestions for countermeasures to backdoored PRGs, which we call *immunizers*. We show that simply hashing PRG outputs is not an effective immunizer against an attacker that knows the hash function in use. Salting the hash, however, does yield a secure immunizer, a fact we prove using a surprisingly subtle proof in the random oracle model. We also give a proof in the standard model under the assumption that the hash function is a universal computational extractor (a recent notion introduced by Bellare, Tung, and Keelveedhi).

1 Introduction

Pseudorandom number generators (PRGs) stretch a short, uniform bit string to a larger sequence of pseudorandom bits. Beyond being a foundational primitive in cryptography, they are used widely in practice within applications requiring relatively large amounts of cryptographic randomness. Seed the PRG via the

output of some (more expensive to use) source of randomness, such as a system random number generator, and then use it to efficiently generate effectively unbounded number of pseudorandom bits for the application. Unfortunately, an adversary that can distinguish such bits from uniform or, worse yet, outright predict the outputs of a PRG, almost invariably compromises security of higher level applications. This fragility in the face of poor pseudorandom sources is borne out by a long history of vulnerabilities [7, 8, 14, 16, 17, 22, 24, 33].

Perhaps it is no coincidence, then, that PRGs have also been a target for backdoors. As far back as 1983, Vazirani and Vazirani [30, 31] introduce the notion of trapdoored PRGs and show the Blum-Blum-Shub PRG is one [10]. Their purpose was not for sabotaging systems, however, but instead they used the property constructively in a higher level protocol. The best known example of potential sabotage is the backdoored NIST Dual EC PRG [23]. It is parameterized by two elliptic curve points; call them P and Q . The entity that selects these points can trivially know $d = \text{dlog}_Q P$, and armed with d any attacker can from an output of the PRG predict all future outputs. This algorithm and the proposed use of it as a way of performing key escrow was detailed at length in a patent by Brown and Vanstone [11]. The possibility of the Dual EC PRG having been standardized so as to include a backdoor was first discussed publicly by Shumow and Ferguson [27]. More recent are allegations that the United States government did in fact retain trapdoor information for the P and Q constants mandated by the NIST standard. The practical implications of this backdoor, should those constants be available, were recently explored experimentally by Checkoway et al. [13]: they quantified how saboteurs might decrypt TLS sessions using the trapdoor information and sufficient computational resources.

Given the importance of backdoored PRGs (and protecting against them), we find it striking that there has been, thus far, no formal treatment of the topic of maliciously backdoored PRGs. We rectify this, giving appropriate notions for backdoored PRGs (building off of [30]) that not only capture Dual EC, but allow us to explore other possible avenues by which a backdoored PRG might be designed, the relationships between this primitive and others, and the efficacy of potential countermeasures against backdoors. We provide an overview of each set of contributions in turn.

Backdoored PRGs. We focus on families of PRGs, meaning that one assumes a parameter generation algorithm that outputs a public set of parameters that we will call, for reasons that will become clear shortly, a public key. A generation algorithm takes a public key, the current state of the generator, and yields a (hopefully) pseudorandom output, as well as a new state. This is standard. A backdoored PRG, on the other hand, has a parameter generation algorithm that additionally outputs a trapdoor value that we will also call a secret key. A backdoored PRG should provide, to any party that has just the public key, a sequence of bits that are indistinguishable from random. To a party with the secret key these bits may be easily distinguishable or, better yet from the attacker’s perspective, predictable with some reasonable success probability.

As an example, the generation algorithm for backdoored Dual EC picks a fixed group element Q , a random exponent d , and outputs as public key the pair $P = Q^d$ and Q . The secret key is d . (We use multiplicative notation for simplicity.) Generation works by taking as input an initial, random state s , and then computing $s' = P^s$ as the next state. An output is computed as all but the last 16 bits of $Q^{s'}$. (We ignore here for simplicity the possible use of additional input.) An attacker that knows d can guess the unknown 16 bits of $Q^{s'}$ and compute $P^{s'}$, the value which defines the next output as well as all future states. In practice applications allow the attacker to check guesses against future PRG outputs, allowing exact discovery of the future state (c.f., [13]).

The Dual EC PRG does not provide outputs that are provably indistinguishable from random bits, and in fact some analysis has shown that despite dropping the low 16 bits, abusable biases remain [25]. A natural question is whether one can build a similarly backdoored PRG but which is provably secure as a PRG?

We answer this in the positive. To do so, we first show a more general result: the equivalence of pseudorandom public-key encryption (PKE) and backdoored PRGs whose outputs are pseudorandom (to those without the trapdoor). Pseudorandom PKE schemes have ciphertexts that are indistinguishable from random bits. Constructions from elliptic curves include Möller’s [21] and the more recent Elligator proposal [9] and its variants [2, 29]. Another approach to achieve pseudorandom bits is via public-key steganography [3, 12, 32, 36]. We give a black-box construction of backdoored PRGs from any pseudorandom PKE. To complete the equivalence we show how any secure backdoored PRG can be used to build pseudorandom PKE scheme. The latter requires using an amplification result due to Holenstein [18].

We also show how a saboteur can get by with key encapsulation mechanisms that have pseudorandom ciphertexts (which are simpler than regular PKE). A KEM encapsulate algorithm takes as input randomness and a public key, and outputs a ciphertext and a one-time-use secret key. We use this algorithm directly as a generator for a backdoored PRG: the ciphertext is the output and the session key is the next state. The secret key for decapsulation reveals the next state. Seen in this light, the Dual EC PRG is, modulo the bit truncations, an instantiation of our generic KEM construction using the ElGamal KEM.

The types of backdoored PRGs discussed thus far only allow use of a trapdoor to predict future states. We formalize another type of backdoored PRG which requires the attacker to be able to determine any output (as chosen at random from a sequence of outputs) using another output (again chosen at random from the same sequence). Such “random access” could be useful to attackers that want to predict previous outputs from future ones, for example.

Immunization countermeasures. So far we have formalized the problem and discussed improved backdoored PRGs. We now turn to countermeasures, a topic of interest given the reduced trust in PRGs engendered by the possibility of backdooring. While the best countermeasure would be to use only trusted PRGs, this may not be possible in all circumstances. For example, existing proprietary software or hardware modules may not be easily changed, or PRG choices may

be mandated by standards, as in the case of FIPS. Another oft-suggested route, therefore, is to efficiently post-process the output of a PRG in order to prevent exploitation of the backdoor. We call such a post-processing strategy an *immunizer*.

A clear candidate for an immunizer is a cryptographic hash function, such as SHA-256 (or SHA-3). A natural assumption is that hashing the output of a PRG will provide security even when the attacker knows the trapdoor, as the hash will hide the data the attacker might use to break PRG security. (This assumption presumes that SHA-256 is itself not backdoored; we have no evidence otherwise, although see [1].) Another, similar idea is to truncate a large number of the output bits.

We show that successful immunization is, perhaps surprisingly, more subtle than naïve approaches like this would suggest. We show that, *a saboteur that knows the immunizer strategy ahead of time can build a backdoored PRG that bypasses the immunizer*. We refer to this setting as the *public immunizer* security model, as both the PRG designer and the backdoor exploiter know the exact immunizer function. We show that for *any* such immunizer, the attacker can leak secret state bit-by-bit. Hence, this is true even when hashing and truncating, and even when modeling the hash function as a random oracle (RO).

This observation suggests that a the designer of a secure PRG should not have exact knowledge of the immunizer. We introduce two further security models for immunizers. In the *semi-private* model, the immunizer can use randomness unknown to the PRG designer, but which is revealed to the backdoor exploiter. In the *private* model, the randomness is never revealed to the saboteur. Constructing provably strong immunizers is straightforward in this last model, but not necessarily practical ones.

For semi-private immunizers, one can prevent basic immunizer-bypassing attacks against hashing (such as we describe below) by using the immunizer's randomness as a salt for the hash. While this immunization strategy thwarts such attacks, proving that it is secure — meaning that its outputs are provably indistinguishable from random bits even for an attacker that has the trapdoor secret of the original backdoored RNG *and the immunization salt* — is surprisingly tricky. One would like to argue that since the PRG must be indistinguishable from random to attackers without the secret trapdoor, then they must have high entropy, and hence hashing with a salt can extract uniform bits from these unpredictable outputs. However, the distinguisher here *does* know the trapdoor, and thus we cannot directly use the assumed backdoored PRG's security against distinguishers who *do not* know the trapdoor. Giving an analysis in the RO model (ROM), we overcome this hurdle by exploiting the fact that, to achieve standard PRG security (no trapdoor available) across multiple invocations with fresh seeds, the backdoored PRG must have low collision probability of outputs. We can in turn use the collision probability as a bound on the predictability of outputs by an adversary, and thereby prove the security of the hashed outputs. We also extend this result to work in the standard model assuming only that the hash function is a universal computational extractor (UCE) [4].

Further related work. As already mentioned, Vazirani and Vazirani [30, 31] introduce the notion of trapdoored generators and use them constructively in protocol design. We build on their notion, but require stronger security in normal operation (indistinguishability from random bits). We also generalize to other trapdoor exploitation models, and study broader connections and countermeasures. Their trapdoor PRG using Blum-Blum-Shub can be recast to work as a backdoored PRG using our KEM-style framework (the generated parity bits being the next state and the final squaring of the seed being the generator output). This approach does produce an unbounded number of bits, however, as no further bits can be produced once the final squaring is output.

Young and Yung studied what they called kleptography: subversion of cryptosystems by modifying encryption algorithms in order to leak information subliminally [34–36]. Juels and Guajardo [20] propose an immunization scheme for kleptographic key-generation protocols that involves publicly-verifiable injection of private randomness by a trusted entity. More recent work by Bellare, Paterson, and Rogaway [5] treats a special case of Young and Yung’s setting for symmetric encryption. We treat a different case, that of PRGs, that has not yet been extensively treated (but our general setting is the same).

Goh et al. [15] investigate how to modify TLS or SSH implementations in order to leak session keys to network attackers that know a trapdoor. One could use a backdoored PRG to accomplish this; indeed this was seemingly the intent behind use of Dual EC in TLS [13]. However, their work does not try to subvert PRGs.

Some of our results, in particular the backdoored PRG that foils public immunizers, use channels that can be viewed as subliminal in the sense introduced by Simmons [28]. Our technique is also reminiscent of the one used to build secret-key steganography [19].

2 Models and Definitions

Notation. We denote the set of all binary strings of length n by $\{0, 1\}^n$, and the set of all binary strings $\{0, 1\}^* = \bigcup_{i=0}^{\infty} \{0, 1\}^i$. We denote the concatenation of two bit strings s_1 and s_2 by $s_1 \| s_2$. We use lsb and lsb_2 to mean the last bit and the last two bits of a bit string, respectively. We denote by $R^{\gg 1}$ and $R^{\gg 2}$ the bit strings obtained by one and two right shifts of R , respectively.

An algorithm is a function mapping inputs from some domain to outputs in some range. For non-empty sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$, we denote the composition of algorithms $F: \mathcal{X} \rightarrow \mathcal{Y}$ and $G: \mathcal{Y} \rightarrow \mathcal{Z}$ by $F \circ G$, i.e. $(F \circ G)(s) = F(G(s))$. A randomized algorithm is an algorithm with a designated bit-string input (always the last) called the coins. We write $F(x; r)$ to denote the output resulting from running F on input x and coins r . We write $y \leftarrow F(x; r)$ to assign y that value. We will write $F(x)$ when the coins are understood from context and write $y \leftarrow F(x)$ to denote picking a fresh r appropriately and running $F(x; r)$. We assume r is always of some sufficient length that we leave implicit. For brevity, we often introduce algorithms without their domain and range when these are clear from context.

The running time of an algorithm is the worst-case number of steps to compute it in an abstract model of computation with unit basic operation costs. In most cases the implementation will be clear, and we will clarify when not; our results extend in straightforward ways to finer-grained models of computation.

We write $x \leftarrow_{\$} \mathcal{X}$ to denote sampling a value x uniformly from a set \mathcal{X} . We use the same notation for non-uniform distributions, and in such cases specify the distribution. We let \mathcal{U}_n denote the uniform distribution over $\{0, 1\}^n$ and \mathcal{U}_n^q the uniform distribution over $\mathcal{U}_n \times \dots \times \mathcal{U}_n$ (q repeats of \mathcal{U}_n). For ease of notation, we abbreviate \mathcal{U}_n to \mathcal{U} when the length n is clear from context. Applying an algorithm (or other function) to a distribution, e.g., $F(x; \mathcal{U})$, denotes the implied distribution over outputs.

PRFs, PRPs, and Encryption. We recall a number of standard cryptographic primitives.

Definition 1 (Computational Indistinguishability). Two distributions X and Y are called (t, ε) -computationally indistinguishable (denoted by $\mathbf{CD}_t(X, Y) \leq \varepsilon$) if for any algorithm D running in time t , $|\Pr[D(X) = 1] - \Pr[D(Y) = 1]| \leq \varepsilon$.

Definition 2 (Pseudorandom Function). A family of algorithms $\{F_{sk}: \{0, 1\}^m \rightarrow \{0, 1\}^n \mid sk \in \{0, 1\}^k\}$ is called a family of (t, q, δ) -pseudorandom functions if $\text{Adv}_F^{\text{PRF}} \triangleq \max_D \text{Adv}_F^{\text{PRF}}(D) \triangleq \max_D (2 |\Pr[\mathcal{G}_F^{\text{PRF}}(D) \Rightarrow \text{true}] - \frac{1}{2}|) \leq \delta$ where the maximum is taken over all algorithms D running in time t and making up to q queries to the oracle \mathcal{O} (the game $\mathcal{G}_F^{\text{PRF}}(D)$ is shown in Fig. 1). Function \mathcal{F} in Fig. 1 is a uniformly selected random function $\mathcal{F}: \{0, 1\}^m \rightarrow \{0, 1\}^n$.

Definition 3 (Pseudorandom Permutation). A family of functions $\{f_{\text{seed}}: \{0, 1\}^n \rightarrow \{0, 1\}^n \mid \text{seed} \in \{0, 1\}^\ell\}$ is called a (t, q, ε) -pseudorandom permutation if it is a (t, q, ε) -pseudorandom function and f_{seed} is a permutation for every $\text{seed} \in \{0, 1\}^\ell$.

Conventional public-key encryption (PKE) schemes meet semantic security style notions, meaning no partial information about plaintexts is leaked. Many traditional ones additionally are such that ciphertexts are indistinguishable from uniformly chosen group elements (e.g., ElGamal). We use something slightly different still: public-key encryption (PKE) with pseudorandom ciphertexts. These schemes have ciphertexts that are indistinguishable from random *bit strings* (of appropriate length). Both theoretical and practical constructions of such public key encryption schemes were shown in [3, 12, 32]. Constructions from elliptic curves include [21] and [9].

Definition 4 (IND\$-CPA Public Key Encryption). A triple $(K, \text{Enc}_{pk}, \text{Dec}_{sk})$, where $K \rightarrow \{0, 1\}^p \times \{0, 1\}^k$, $pk \in \{0, 1\}^p$, $\text{Enc}_{pk}: \{0, 1\}^m \times \{0, 1\}^p \rightarrow \{0, 1\}^n$, $sk \in \{0, 1\}^k$, $\text{Dec}_{sk}: \{0, 1\}^n \rightarrow \{0, 1\}^m$ is called a (t, q, δ) – IND\$-CPA public key encryption scheme if

Game $\mathcal{G}_F^{PRF}(D)$

$$\frac{}{sk \leftarrow \mathcal{U}_k}$$

$$b \leftarrow \{0, 1\}$$

$$\text{if } b = 1 \text{ then}$$

$$| \quad \mathcal{O} \leftarrow F_{sk}$$

$$\text{else}$$

$$| \quad \mathcal{O} \leftarrow \mathcal{F}$$

$$b' \leftarrow D^{\mathcal{O}}$$

$$\text{return } (b = b')$$

Game $\mathcal{G}_{K, Enc}^{CPA}(D)$

$$\frac{}{(pk, sk) \leftarrow \mathbb{S} K}$$

$$b \leftarrow \{0, 1\}$$

$$\text{if } b = 1 \text{ then}$$

$$| \quad \mathcal{O} \leftarrow Enc_{pk}$$

$$\text{else}$$

$$| \quad \mathcal{O} \leftarrow \mathcal{R}$$

$$b' \leftarrow D^{\mathcal{O}}(pk)$$

$$\text{return } (b = b')$$
Fig. 1. PRF game**Fig. 2.** IND\$-CPA Game

- $\Pr[\text{Dec}_{sk}(\text{Enc}_{pk}(s; \alpha)) = s] = 1$, where $s \leftarrow \{0, 1\}^m$, $(pk, sk) \leftarrow K$, $\alpha \leftarrow \{0, 1\}^\rho$,
- $\text{Adv}_{K, Enc}^{\text{CPA}}(D) \triangleq 2 |\Pr[\mathcal{G}_{K, Enc}^{\text{CPA}}(D) \Rightarrow \text{true}] - \frac{1}{2}| \leq \delta$ for any algorithm D running in time t and making up to q queries to the oracle \mathcal{O} . (The game $\mathcal{G}_{K, Enc}^{\text{CPA}}(D)$ is defined in Fig. 2, the function \mathcal{R} outputs a uniformly selected output of length n .)

Pseudorandom generators. A pseudorandom generator (PRG) is a pair of algorithms (K, G) . The parameter generation algorithm K takes input coins and outputs a pair (pk, sk) , called the public key and secret or private key (or trapdoor). Traditionally, a PRG has no trapdoor, and pk would be referred to as the public parameter. Our notation of public / private keys is for consistency with the next section; for an ordinary PRG, sk may be taken as null. We assume that sk uniquely determines pk . A public key pk designates a family of algorithms denoted by G . Each algorithm $G_{pk} : \mathcal{S} \rightarrow \{0, 1\}^n \times \mathcal{S}$ maps an input called the state to an n -bit output and a new state. We drop the subscript pk where it is clear from context. We refer to \mathcal{S} as the state space; it will often simply be bit strings of some length. We will always specify a distribution over \mathcal{S} that specifies the selection of an initial state, written $s \leftarrow \mathbb{S} \mathcal{S}$, for the PRG. For any integer $q \geq 1$, we let $\text{out}^q(G, s)$ for $s \in \mathcal{S}$ denote the sequence of bit strings (r_1, r_2, \dots, r_q) output by running $(r_1, s_1) \leftarrow G(s)$, then $(r_2, s_2) \leftarrow G(s_1)$, and so on. By $\text{state}^q(G, s)$ we denote the sequence of states (s_1, s_2, \dots, s_q) . A PRG is secure when no adversary can distinguish between its outputs and random bits.

Definition 5 (PRG security). A PRG (K, G) is a (t, q, δ) -secure PRG if for $pk \leftarrow K$, $\text{CD}_t((pk, \text{out}^q(G_{pk}, \mathcal{U})), \mathcal{U}) \leq \delta$.

This definition does not capture forward-secrecy, meaning that past outputs should be indistinguishable from random bits even if the current state is revealed. In all the PRG constructions that follow, we point out which of the results satisfy the forward-security notion and which are forward-insecure.

3 Backdoored Pseudorandom Generators

A backdoored pseudorandom generator (BPRG) is a triple of algorithms (K, G, A) . The pair (K, G) is a PRG, as per the definition in the last section. The third algorithm A we call the adversary, although it is in fact co-designed with the rest of the scheme. It uses the trapdoor output by K to violate security of the PRG in one of several potential ways. We give games defining these distinct ways of violating security in Figure 3.

Game $\mathcal{G}_{\text{dist}}^{\text{BPRG}}(K, G, A)$	Game $\mathcal{G}_{\text{next}}^{\text{BPRG}}(K, G, A)$	Game $\mathcal{G}_{\text{rseek}}^{\text{BPRG}}(K, G, A, i, j)$
$(pk, sk) \leftarrow \$ K$	$(pk, sk) \leftarrow \$ K$	$(pk, sk) \leftarrow \$ K$
$s \leftarrow \$ \mathcal{S}$	$s \leftarrow \$ \mathcal{S}$	$s \leftarrow \$ \mathcal{S}$
$r_1^0, \dots, r_q^0 \leftarrow \text{out}^q(G_{pk}, s)$	$r_1, \dots, r_q \leftarrow \text{out}^q(G_{pk}, s)$	$r_1, \dots, r_q \leftarrow \text{out}^q(G_{pk}, s)$
$r_1^1, \dots, r_q^1 \leftarrow \$ \mathcal{U}_n^q$	$s_1, \dots, s_q \leftarrow \$ \text{state}^q(G_{pk}, s)$	$r_j' \leftarrow \$ A(sk, i, j, r_i)$
$b \leftarrow \$ \{0, 1\}$	$s'_q \leftarrow \$ A(sk, r_1, \dots, r_q)$	return $(r_j = r_j')$
$b' \leftarrow A(sk, r_1^b, \dots, r_q^b)$	return $(s'_q = s_q)$	
return $(b = b')$		

Fig. 3. Security games defining success of trapdoor-equipped adversaries

The first game is identical to the standard PRG definition except that the adversary here gets the trapdoor. The second tasks A with recovering the current state, given the trapdoor and a sequence of outputs. This is, by definition, sufficient information to produce all future outputs of G_{pk} . The last tasks A with predicting the full output of some state j given the trapdoor and the output for i .

Definition 6 (Backdoored PRG). A triple (K, G, A) is called a $(t, q, \delta, (\mathcal{G}_{\text{type}}, \epsilon))$ -backdoored PRG for $\text{type} \in \{\text{dist}, \text{next}, \text{rseek}\}$ if (K, G) is a (t, q, δ) -secure PRG and $\text{Adv}_{\text{type}}^{\text{BPRG}}(K, G, A) \geq \epsilon$, where

$$\begin{aligned} \text{Adv}_{\text{dist}}^{\text{BPRG}}(K, G, A) &\triangleq 2 \cdot \left| \Pr[\mathcal{G}_{\text{dist}}^{\text{BPRG}}(K, G, A) \Rightarrow \text{true}] - \frac{1}{2} \right|, \\ \text{Adv}_{\text{next}}^{\text{BPRG}}(K, G, A) &\triangleq \Pr[\mathcal{G}_{\text{next}}^{\text{BPRG}}(K, G, A) \Rightarrow \text{true}], \text{ and} \\ \text{Adv}_{\text{rseek}}^{\text{BPRG}}(K, G, A) &\triangleq \min_{1 \leq i, j \leq q} \Pr[\mathcal{G}_{\text{rseek}}^{\text{BPRG}}(K, G, A, i, j) \Rightarrow \text{true}]. \end{aligned}$$

A $\mathcal{G}_{\text{dist}}$ -BPRG is only interesting when $\epsilon \gg \delta$, as otherwise the distinguisher without the trapdoor information can distinguish already with advantage δ . For the other types, even if $\epsilon < \delta$ the definition is still meaningful.

A $(t, q, \delta, (\mathcal{G}_{\text{next}}, \epsilon))$ -BPRG is (strictly) better for the saboteur than achieving a $\mathcal{G}_{\text{dist}}$ -BPRG under the same parameters. The random seek notion is orthogonal; it may or may not be better depending on the situation. Our attacks and

(looking ahead to later sections) defenses will be given according to the strongest definitions. That is when taking on the role of the saboteur, we will build $\mathcal{G}_{\text{next}}$ -BPRGs and/or $\mathcal{G}_{\text{seek}}$ -BPRGs with as efficient as possible A. When considering defenses against saboteurs by way of immunization, we will target showing that no efficient A can succeed in $\mathcal{G}_{\text{dist}}$.

Example: the Dual EC BPRG. As an example of a BPRG we turn to Dual EC. It uses an elliptic curve group \mathbb{G} with generator g . For consistency with later sections, we use multiplicative notation for group operations. We also skip for simplicity some details about representation of elliptic curve points, these being unimportant for understanding the attack. For a more detailed description of the algorithm and backdoor see [13].

Key generation K picks a random point $Q \in \mathbb{G}$ and an exponent $d \leftarrow \mathbb{Z}_{|\mathbb{G}|}$. It computes $P = Q^d$. The public key is set to $pk = (P, Q)$ and the secret is x . The state space is $\mathcal{S} = \mathbb{Z}_{|\mathbb{G}|}$. On input a seed $s_i \in \mathcal{S}$, the generation algorithm G computes $s_{i+1} \leftarrow P^{s_i}$ and computes r_{i+1} as all but the last 16 bits of $Q^{s_{i+1}}$. The output is (r_{i+1}, s_{i+1}) .

With knowledge of d and given two consecutive outputs r_1, r_2 corresponding to states s, s_1 we can give a $\mathcal{G}_{\text{next}}$ adversary A that efficiently recovers s_2 . Adversary A starts by computing from r_1 a set of at most 2^{16} possibilities for Q^{s_1} . Let these possibilities be $X_1, \dots, X_{2^{16}}$. Then for each $i \in [1..2^{16}]$, the adversary checks whether $Q^{X_i^d}$ has all but last 16 bits that match r_2 . If so it outputs $s_2 = X_i^d = Q^{s_1 d} = P^{s_1}$. Note that while A cannot recover the generator's second state s_1 , it can predict the generator's second output r_2 , the third state s_2 , and all subsequent states and outputs. Also A is relatively efficient, working in time about 2^{16} operations.

As for basic PRG security without the trapdoor, a result due to Schoenmakers and Sidorenko [25] gives an attack working in time about 2^{16} using a single output to achieve distinguishing advantage around $1/100$. Thus, putting it all together, we have that Dual EC is a $(t, q, \delta, (\mathcal{G}_{\text{next}}, 1))$ -BPRG for $t \approx 2^{16}$, $q > 2$, and $\delta \approx 1/100$.

From a saboteur's perspective, that Dual EC doesn't achieve PRG security (against distinguishers without the trapdoor) seems a limitation. One can truncate more than 16 bits to achieve better security, but this would make A exponentially less efficient. In the next section we will show how a saboteur can construct a BPRG with strong PRG security and efficient state recovery.

4 Backdoored PRG Constructions

We start by simplifying and improving the Dual EC BPRG. Let G be a group and g a generator of G . Let K pick a random secret key $x \leftarrow \mathbb{Z}_{|G|}$ and let $pk \triangleq X = g^x$. The PRG works simply as $G(pk, s_i) = (r_{i+1}, s_{i+1}) = (g^{s_i}, X^{s_i})$. A $\mathcal{G}_{\text{next}}$ adversary can recover $s_{i+1} = X^{s_i}$ by computing r_{i+1}^x . For a G that is DDH secure and for which uniform group elements are indistinguishable from bit strings (e.g., [2, 9, 29]), this construction can be proven $\mathcal{G}_{\text{dist}}^{\text{PRG}}$ secure under the DDH assumption.

\underline{K}'	$\frac{G'(pk, s)}{(r', s') \leftarrow \text{Encap}(pk; s)}$	$\frac{A'(sk, r_1, \dots, r_q)}{s' \leftarrow \text{Decap}(sk, r_q)}$
return (pk, sk)	return (r', s')	return s'

Fig. 5. Backdoored PRG from a pseudorandom KEM

4.1 Backdoored PRGs from Key Encapsulation

We in fact can generalize significantly by observing that the previous construction is actually using the ElGamal key encapsulation scheme (KEM) in a direct way. Recall that a KEM scheme triple of algorithms is a KEM $\Gamma = (\text{Gen}, \text{Encap}, \text{Decap})$. The key generation outputs a public key / secret key pair $(pk, sk) \leftarrow \text{Gen}$. The encapsulation algorithm takes the public key, random coins $r \in \{0, 1\}^n$ for some n and outputs a ciphertext-key pair $(c, K) \leftarrow \text{Encap}(pk; r)$ where $K \in \{0, 1\}^n$. The decapsulation algorithm takes a secret key and ciphertext and outputs a key: $\text{Decap}(sk, c) = \tilde{K} \in \{0, 1\}^n \cup \{\text{invalid}\}$. We require correctness, meaning that $\text{Decap}(sk, c) = K$ for $(c, K) = \text{Encap}(pk; r)$ and for all pk, sk pairs generatable by Gen and all coin strings r .

We give a variant of KEM security that requires ciphertexts to be pseudorandom: the output of Encap is indistinguishable from a pair of random bit strings. See Figure 4. We define $\text{Adv}_{\text{KEM}}^{\text{Dist}}(\mathcal{D}) = 2|\Pr[\text{Dist}_{\text{KEM}}^{\mathcal{D}} \Rightarrow \text{true}] - \frac{1}{2}|$. A KEM Γ is said to be a (t, δ) -pseudorandom KEM if $\text{Adv}_{\Gamma}^{\text{Dist}} := \max_{\mathcal{D}} \text{Adv}_{\Gamma}^{\text{Dist}}(\mathcal{D}) \leq \delta$, where the maximum is taken over all algorithms \mathcal{D} running in time t .

This is a strictly stronger security notion than the conventional one for KEMs [26], which does not demand that ciphertexts have any particular appearance to attackers. This stronger pseudorandomness requirement was first introduced by Möller [21]. He gave an elliptic curve variant of ElGamal that provably meets it, and other KEM constructions can be built using [2, 9, 29].

We have the following result showing that any pseudorandom KEM gives a $\mathcal{G}_{\text{next}}$ -BPRG.

Proposition 1. *Let $\Gamma = (\text{Gen}, \text{Encap}, \text{Decap})$ be a (t, δ) -pseudorandom KEM. Then (\underline{K}', G', A') defined in Fig. 5 is a $(t, q, q\delta, (\mathcal{G}_{\text{next}}, 1))$ -BPRG.*

Game $\text{Dist}_{\text{KEM}}^{\mathcal{D}}$
$(pk, sk) \leftarrow \text{Gen}$
$r \leftarrow \mathbb{S} \{0, 1\}^n$
$(c_0, K_0) \leftarrow \mathbb{S} \text{Encap}(pk; r)$
$c_1 \leftarrow \mathbb{S} \{0, 1\}^n$
$K_1 \leftarrow \mathbb{S} \{0, 1\}^n$
$b \leftarrow \mathbb{S} \{0, 1\}$
$b' \leftarrow \mathbb{S} \text{D}(pk, c_b, K_b)$
return $(b = b')$

Fig. 4. Pseudorandom KEM security

Proof. The correctness of the KEM gives that $\text{Adv}_{\text{next}}^{\text{BPRG}}(K', G', A') = 1$ and that A' is efficient. We now prove (standard) PRG security against distinguishers without the trapdoor. To do so we use a hybrid argument, applying the pseudorandom KEM security q times. Let H_0 be the $\mathcal{G}_{\text{dist}}^{\text{PRG}}$ game with $b = 0$ and H_q be $\mathcal{G}_{\text{dist}}^{\text{PRG}}$ with $b = 1$. Let H_i for $1 \leq i \leq q - 1$ be the same as game H_{i-1} except that we replace the i^{th} output of Encap with two independent, random bit strings. A straightforward reduction gives that $\mathbf{CD}_t(H_i, H_{i+1}) \leq \delta$, and since we have q hybrids $\text{Adv}_{\text{dist}}^{\text{PRG}}(K, G, D) \leq q\delta$ for any D running in time t . \square

4.2 Random Seek Backdoored PRGs

We now show a prediction attack, where the prediction algorithm can *seek* to any output quickly. Given one output, we can predict any other, and the prediction can seek in both directions, that is predict previous outputs as well. In the construction shown, we use the lsb of a random string to make a decision, and we shift by one bit, so that the randomness used later is independent of the bit used for decision. We assume that the underlying PRG or PRF was used to get enough number of bits so that after the shift we have enough random bits for encryption.

Proposition 2. *Let $(K, \text{Enc}_{pk}, \text{Dec}_{sk})$ be a (t, q, δ) -IND\$-CPA public key encryption scheme, F_{sk} be a (t, q, δ) -pseudorandom function. Then (K', G', A') defined in Fig. 6 is a $(t, q, 3\delta, (\mathcal{G}_{\text{rseek}}, \frac{1}{4} - \delta))$ -backdoored pseudorandom generator.*

K'	$G'(pk, (s_0, s_1, \text{count}))$	$A'(sk, i, j, r_i)$
$(pk, sk) \leftarrow K$	$\alpha \leftarrow F_{s_1}(\text{count})$	if $(i = j)$ then
return (pk, sk)	if $\text{lsb}(\alpha) = 0$ then	return r_i
	$r \leftarrow \text{Enc}_{pk}(s_0; \alpha^{\gg 1})$	$s_0 \leftarrow \text{Dec}_{sk}(r_i)$
	else	if $(s_0 = \perp)$ then
	$r \leftarrow F_{s_0}(\text{count})$	return 0
	$\text{count} \leftarrow \text{count} + 1$	$r_j \leftarrow F_{s_0}(j)$
	return $(r, (s_0, s_1, \text{count}))$	return r_j

Fig. 6. Random seek backdoored PRG

Proof.

$$\text{Adv}_{\text{rseek}}^{\text{BPRG}}(K', G', A') = \Pr[r_j = r'_j] \geq \Pr[\text{lsb}(F_{s1}(i)) = 0 \wedge \text{lsb}(F_{s1}(j)) = 1] \geq \frac{1}{4} - \delta.$$

From pseudorandomness of F 's outputs

$$\mathbf{CD}_t(F_{s0}(1), \dots, F_{s0}(q), \mathcal{U}) \leq \delta, \mathbf{CD}_t(F_{s1}(1), \dots, F_{s1}(q), \mathcal{U}) \leq \delta.$$

Then $\mathbf{CD}_t((pk, s_0, \text{Enc}_{pk}(s_0; \alpha^{\gg 1})), (pk, s_0, \mathcal{U})) \leq 2\delta$ due to IND\$-CPA security. Thus,

$$\mathbf{CD}_t((pk, \text{out}^q(\mathcal{G}'_{pk}, \mathcal{U})), \mathcal{U}) \leq 3\delta.$$

□

The distinguishing and predicting PRGs we discussed also satisfy the notion of forward security, whereas the $\mathcal{G}_{\text{rseek}}$ construction in Fig 6 is forward-insecure.

4.3 Public-Key Encryption from a Backdoor PRG

We show that the existence of backdoored PRGs implies public-key encryption (PKE). From a backdoored PRG, we construct a bit encryption scheme with noticeable correctness and overwhelming secrecy. Using parallel repetition and privacy amplification of key-agreement [18], we can amplify secrecy and correctness without increasing the number of rounds. Since the number of rounds is not increased, we obtain secure public-key encryption.

Theorem 1. *If (K, G_{pk}, A) is a $(t, q, \delta, (\mathcal{G}_{\text{dist}}, \varepsilon))$ -backdoored PRG, then the protocol in Fig. 7 is a bit-encryption protocol with correctness ε and security $1 - \delta$ against attackers running in time t .*

$\underline{\text{Gen}}$ $(pk, sk) \leftarrow K$ return (pk, sk)	$\text{Enc}(pk, b)$ $s \leftarrow \mathcal{U}$ if $(b = 0)$ then $r \leftarrow \mathcal{U}$ else $r \leftarrow \text{out}^q(G_{pk}, s)$ return r	$\text{Dec}(sk, r)$ $b' \leftarrow A(sk, r)$ return b'
---	--	---

Fig. 7. Bit Encryption

Proof. orrectness:

$$\Pr[\text{Dec}(sk, \text{Enc}(pk, b)) = b] = \Pr[b = b'] = \Pr[\mathcal{G}_{\text{dist}}^{\text{BPRG}}(K, G, A) \Rightarrow \text{true}] \geq \frac{1}{2} + \frac{\varepsilon}{2}.$$

Security:

For any adversary D who runs in time t ,

$$\Pr[D(pk, r) = b] = \frac{1 + \mathbf{CD}_t((pk, \text{out}^q(G, \mathcal{U})), \mathcal{U})}{2} \leq \frac{1}{2} + \frac{\delta}{2}.$$

□

Note that combining this result with our earlier construction of backdoored PRGs from PKE and Proposition 1, we arrive at the promised conclusion that backdoored PRGs and pseudorandom PKE are equivalent. We capture this with the following informal theorem, which is a corollary of the results so far.

Theorem 2. *Backdoor PRGs exist iff public-key encryption with pseudorandom ciphertexts exists.*

5 Immunization

In this section, we ask how to *immunize* a potentially backdoored PRG. A natural idea is for the user to apply a non-trivial function f to the output of the PRG. So now the attacker A learns $f(r_i)$ rather than r_i . We ask the question: when does f successfully immunize a PRG? We study the immunization functions that turn a backdoored PRG into a backdoor-less PRG. Letting the immunization function be a family of algorithms $\{f_{\text{seed}} \mid \text{seed} \in \{0,1\}^\ell\}$, we consider the following immunization models:

1. Public immunization: In this model, `seed` is revealed to the attacker A prior to construction of the PRG algorithm. The attacker thus knows the immunization function f_{seed} that will be applied to the outputs of the generator. In this setting, the goal of the attacker A is to develop a PRG G with a backdoor that bypasses the known immunization.
2. Semi-private immunization: In this model, the PRG generator G is constructed without reference to `seed`. We may view this as a setting in which the PRG attacker A learns `seed`, and thus f_{seed} , only *after* the specification of G. This situation can arise, for example, when the immunization function f depends upon a source of fresh public randomness.
3. Private immunization: In this model, `seed` is secret, in the sense that G is constructed without reference to `seed` and A never learns `seed`. We might imagine the user using a source of private randomness, unavailable to A, to seed the immunization function f . (Note that although the user has some private randomness, she might still need the PRG to generate longer pseudorandom strings.)

Now we give formal definitions of secure immunization in the three models discussed above. We slightly abuse notation in the following way: For a PRG G such that $(r_i, s_i) \leftarrow G(s_{i-1})$, we write $f \circ G$ to mean $f(r_i)$, i.e., f applied to the output of G only (and not G's internal state). Similarly, by $\text{out}^q(f \circ G, s)$ we mean the sequence $(f(r_1), \dots, f(r_q))$, where $(r_1, \dots, r_q) = \text{out}^q(G, s)$.

Definition 7 (Public Immunization). *Let $\text{type} \in \{\text{dist}, \text{next}, \text{rseek}\}$. A family of algorithms $\{f_{\text{seed}} \mid \text{seed} \in \{0,1\}^\ell\}$ is called a public $((t, q, \delta), (t', q', \delta'), (\mathcal{G}_{\text{type}}, \varepsilon))$ -immunization, if for any (t, q, δ) -secure PRG (K, G) , and for any algorithm A running in time t' ,*

- $\{f_{\text{seed}} \circ G_{pk}(\text{seed}, \cdot) \mid (\text{seed}, pk) \in \{0,1\}^\ell \times \{0,1\}^p\}$ is a (t', q', δ') -pseudorandom generator,

$$- \text{Adv}_{\text{type}}^{\text{BPRG}}(K, f_{\text{seed}} \circ G(\text{seed}, \cdot), A(\text{seed}, \cdot)) \leq \varepsilon.$$

Definition 8 (Semi-private Immunization). Let $\text{type} \in \{\text{dist}, \text{next}, \text{rseek}\}$. A family of algorithms $\{f_{\text{seed}} \mid \text{seed} \in \{0,1\}^\ell\}$ is called a semi-private $((t, q, \delta), (t', q', \delta'), (\mathcal{G}_{\text{type}}, \varepsilon))$ -immunization, if for any (t, q, δ) -secure PRG (K, G) , and for any algorithm A running in time t' ,

- $\{f_{\text{seed}} \circ G_{pk}(\cdot) \mid (\text{seed}, pk) \in \{0,1\}^\ell \times \{0,1\}^p\}$ is a (t', q', δ') -pseudorandom generator;
- $\text{Adv}_{\text{type}}^{\text{BPRG}}(K, f_{\text{seed}} \circ G(\cdot), A(\text{seed}, \cdot)) \leq \varepsilon$.

Definition 9 (Private Immunization). Let $\text{type} \in \{\text{dist}, \text{next}, \text{rseek}\}$. A family of algorithms $\{f_{\text{seed}} \mid \text{seed} \in \{0,1\}^\ell\}$ is called a private $((t, q, \delta), (t', q', \delta'), (\mathcal{G}_{\text{type}}, \varepsilon))$ -immunization, if for any (t, q, δ) -secure PRG (K, G) , and for any algorithm A running in time t' ,

- $\{f_{\text{seed}} \circ G_{pk}(\cdot) \mid (\text{seed}, pk) \in \{0,1\}^\ell \times \{0,1\}^p\}$ is a (t', q', δ') -pseudorandom generator;
- $\text{Adv}_{\text{type}}^{\text{BPRG}}(K, f_{\text{seed}} \circ G(\cdot), A(\cdot)) \leq \varepsilon$.

In Section 5.1 we show that it is possible to successfully create a PRG backdoor in the public immunization setting. On the other hand, in Section 5.2 we show that there exist immunizations in the semi-private model that separate these two models. Also, as we will see in Section 5.3, a pseudorandom function is a secure private immunization. To separate semi-private and private models, in the following simple lemma we show that a pseudorandom permutation is not a semi-private immunization. The construction we give for the separation in Fig. 8 also satisfies forward security.

Lemma 1. Let f_{seed} be a (t, q, δ) -pseudorandom permutation. Then there exists a triple (K', G', A') , such that $(K', f_{\text{seed}} \circ G'(\cdot), A'(\text{seed}, \cdot))$ is a $(t, q, 2q\delta, (\mathcal{G}_{\text{next}}, 1))$ -backdoored pseudorandom generator.

Proof. Let G be a (t', q, δ) -pseudorandom generator, and $\Gamma = (\text{Gen}, \text{Encap}, \text{Decap})$ be a (t, δ) -pseudorandom ciphertext KEM. Consider the triple $(K', G'_{pk}, A'(\text{seed}, \cdot))$ shown in Fig. 8. Then $\text{Adv}_{\text{next}}^{\text{BPRG}}(K', f_{\text{seed}} \circ G', A') = \Pr[\text{Decap}(r', sk) = s' \mid (r', s') \leftarrow \text{Encap}(pk; \alpha), (pk, sk) \leftarrow \text{Gen}] = 1$. From pseudorandomness of G's outputs $\mathbf{CD}_t(\text{out}^q(G, \mathcal{U}), \mathcal{U}) \leq \delta$, (t, δ) -pseudorandomness of Γ , and using a hybrid argument similar to the proof of Proposition 1,

$$\mathbf{CD}_t((pk, \text{out}^q(G'_{pk}, \mathcal{U})), \mathcal{U}) \leq 2q\delta.$$

□

5.1 Public Immunization Model

In the public immunization model, the PRG algorithms G and A know the seed of the immunization function that will be applied on the output. In this section

$\underline{K'}$	$\underline{G'(pk, s)}$	$\underline{A'(sk, \text{seed}, f_{\text{seed}}(r_i))}$
$(pk, sk) \leftarrow \text{Gen}$	$(\alpha, \beta) \leftarrow G(s)$	$c \leftarrow f_{\text{seed}}^{-1}(f_{\text{seed}}(r_i))$
return (pk, sk)	$(r', s') \leftarrow \text{Encap}(pk; \alpha)$	$s' \leftarrow \text{Decap}(c, sk)$
	return (r', s')	return s'_i

Fig. 8. PRP Immunization Insecure in Semi-private Model

we demonstrate backdoored pseudorandom generator that cannot be immunized in the public immunization model. Since seed of the immunization function f_{seed} is known to both G and A , in order to construct a backdoored pseudorandom generator from the viewpoint of the saboteur, we fix the strongest function from this family so that for any function in the family, the backdoored PRG after immunization is compromised. The idea behind the construction is to leak the initial state bit by bit, by rejection sampling of the output of the immunization function such that the bias is unnoticeable. For a bit string s , we denote the i th bit of s by $s_{(i)}$.

$\underline{K'}$	$\underline{G'(pk, \text{seed}, (s_0, s_1, \text{count}))}$	$\underline{A'(sk, \text{seed}, f_{\text{seed}}(r_1), \dots, f_{\text{seed}}(r_q))}$
$(pk, sk) \leftarrow K$	$c \leftarrow \text{Enc}_{pk}(s_1)$	$\text{for } 1 \leq i \leq L \text{ do}$
return (pk, sk)	$L \leftarrow c $	$ c_i \leftarrow f_{\text{seed}}(r_i)_{(1)}$
	if $\text{count} \leq L$ then	$s_1 \leftarrow \text{Dec}_{sk}(c)$
	$ j \leftarrow 0$	for $L + 1 \leq i \leq q$ do
	$ s'_0 \leftarrow s_0$	$ r'_i, s_1 \leftarrow G(s_1)$
	$ \text{count}_2 \leftarrow 0$	return $(0, s_1, q + 1)$
	repeat	
	$ \text{count}_2 \leftarrow \text{count}_2 + 1$	
	$ (r', s'_0) \leftarrow G(s'_0)$	
	until $(f_{\text{seed}}(r')_{(1)} = c_{(\text{count})}) \vee$	
	$(\text{count}_2 > \frac{\ln L}{1-\delta});$	
	$ s'_1 \leftarrow s_1$	
	else	
	$ (r', s'_1) \leftarrow G(s_1)$	
	$ s'_0 \leftarrow 0$	
	$ \text{count}' \leftarrow \text{count} + 1$	
	return $(r', (s'_0, s'_1, \text{count}'))$	

Fig. 9. Predicting backdoored PRG in Public Model

Lemma 2. Let $(K, \text{Enc}_{pk}, \text{Dec}_{sk})$ be a $(t, q, \delta) - \text{IND\$-CPA}$ public key encryption scheme, G be a (t, q, δ) -pseudorandom generator, and f such that for $\text{seed} \leftarrow \{0, 1\}^\ell$, $\mathbf{CD}_t(f_{\text{seed}}(\mathcal{U}), \mathcal{U}) \leq \delta$. Then (K', G', A') defined in Fig. 9 is a $(t, q - L \frac{\ln L}{1-\delta}, 2\delta, (\mathcal{G}_{\text{next}}, \varepsilon))$ -backdoored pseudorandom generator, where $\varepsilon = 1 - L \cdot \exp\left(\frac{-\ln^2 L}{3(1-\delta)}\right)$, L is the length of ciphertexts produced by Enc_{pk} .

Proof. From pseudorandomness of G 's outputs $\mathbf{CD}_t(\mathsf{out}^q(G, \mathcal{U}), \mathcal{U}) \leq \delta$ and pseudorandomness of ciphertexts $\mathbf{CD}_t((pk, s_1, \text{Enc}_{pk}(s_1)), (pk, s_1, \mathcal{U})) \leq \delta$,

$$\mathbf{CD}_t((pk, \mathsf{out}^q(G'_{pk}, \mathcal{U})), \mathcal{U}) \leq 2\delta.$$

From the Chernoff bound:

$$\text{Adv}_{\text{next}}^{\text{BPRG}}(K', G', A') \geq 1 - L \cdot \Pr[\text{count}_2 \geq \frac{\ln L}{1 - \delta}] \geq 1 - L \cdot \exp\left(\frac{-\ln^2 L}{3(1 - \delta)}\right).$$

□

5.2 Semi-Private Immunization Model

In the semi-private model, the generator G does not know f_{seed} , but the attacker does. We show that a Random Oracle and a Universal Computational Extractor are secure immunizations in the semi-private model. We will first bound the collision probability of pseudorandom outputs. The collision probability bounds the probability that an algorithm can predict the output of a PRG run on a uniformly random seed, even with the knowledge of some trapdoor information, because, intuitively, the output of a PRG should depend on the input seed also.

Definition 10. *The conditional predictability of X conditioned on Y is defined as*

$$\text{Pred}(X|Y) := \mathbb{E}_{y \leftarrow Y} [\max_x (\Pr[X = x|y = Y])].$$

Definition 11. *The conditional collision probability of X conditioned on Y is defined as*

$$\text{Col}(X|Y) := \mathbb{E}_{y \leftarrow Y} [\Pr_{x_1, x_2 \leftarrow X} [x_1 = x_2|Y = y]].$$

Lemma 3. *For any distributions X and Y , $\text{Pred}(X|Y) \leq \sqrt{\text{Col}(X|Y)}$.*

Proof. Let $p_y = \Pr[Y = y]$, $p_{x|y} = \Pr[X = x|Y = y]$. Then

$$\begin{aligned} \text{Pred}[X|Y] &= \sum_y p_y \cdot \max_x p_{x|y} = \sum_y \sqrt{p_y} \cdot (\sqrt{p_y} \max_x p_{x|y}) \leq \\ &\sqrt{\sum_y p_y \cdot \sum_y p_y \max_x p_{x|y}^2} \leq \sqrt{1 \cdot \sum_y (p_y \cdot \sum_x p_{x|y}^2)} = \sqrt{\text{Col}(X|Y)}. \end{aligned}$$

□

Let $\{G_{pk}: \{0,1\}^m \rightarrow \{0,1\}^n \times \{0,1\}^m | pk \in \{0,1\}^p\}$ be a family of algorithms, then by $\mathsf{out}_i(G_{pk}, \mathcal{U})$ we denote the distribution of G_{pk} 's i th output, i.e. the distribution of r_i where $(r_1, \dots, r_i, \dots, r_q) \leftarrow \mathsf{out}^q(G_{pk}, \mathcal{U})$.

Lemma 4. Let $\{G_{pk} : \{0,1\}^m \rightarrow \{0,1\}^n \times \{0,1\}^m | pk \in \{0,1\}^p\}$ be a (t, q, δ) -pseudorandom generator.¹ Then for any $1 \leq i \leq q$, for any $K \rightarrow \{0,1\}^p \times \{0,1\}^k$ such that $\mathbf{CD}_t(pk, U_p) \leq \delta$, where $(pk, sk) \leftarrow K$,

$$\text{Pred}(\text{out}_i(G_{pk}, \mathcal{U})|sk) \leq \sqrt{\delta + \frac{1}{2^n}}.$$

Proof. We show that $\text{Col}(\text{out}_i(G_{pk}, \mathcal{U})|sk) \leq \delta + \frac{1}{2^n}$, then Lemma 3 implies the desired bound.

Assume, to the contrary, $\text{Col}(\text{out}_i(G_{pk}, \mathcal{U})|sk) > \delta + \frac{1}{2^n}$. This implies that there exists i such that $E_{(pk, sk) \leftarrow K} \Pr[r_i = r'_i|sk] > \delta + \frac{1}{2^n}$, where $r_i, r'_i \leftarrow \text{out}_i(G_{pk}, \mathcal{U})$. Let D be a PRG-distinguisher for G_{pk} as defined in Fig. 10. Then,

$$|\Pr[D(\text{out}^q(G_{pk}, \mathcal{U})) = 1] - \Pr[D(U) = 1]| \geq \text{Col}(\text{out}_i(G_{pk}, \mathcal{U})|sk) - \frac{1}{2^n} > \delta,$$

which contradicts the (t, q, δ) -pseudorandomness of $\{G_{pk}\}$. \square

```

D(pk, r1, ..., rq)
  s ← {0, 1}m
  r'1, ..., r'q ← outq(Gpk, s)
  if ri = r'i then
    | return 1
  else
    | return 0

```

Fig. 10. Distinguisher D for G_{pk}

Positive Result in Random Oracle Model. A random oracle (RO) is an oracle that responds to every unique query with a random response chosen uniformly from its output domain. If a query is repeated it responds the same way every time that query is submitted. A RO : $\{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ is chosen uniformly at random from the set of all functions that map $\{0,1\}^{n+k}$ to $\{0,1\}^n$. We show that in the semi-private model, a Random Oracle is a secure immunization function.

Theorem 3. Let $\{G_{pk} : \{0,1\}^m \rightarrow \{0,1\}^n \times \{0,1\}^m | pk \in \{0,1\}^p\}$ be a (t, q, δ) -pseudorandom generator. Then $f_{\text{seed}}(x) = \text{RO}(x||\text{seed})$ is a semiprivate $((t, q, \delta), (t, q, \delta), (\mathcal{G}_{\text{dist}}, \varepsilon))$ -immunization for G_{pk} , where

$$\varepsilon = \delta + \frac{q^2}{2^n} + \frac{q_G}{2^k} + qq_A \sqrt{\delta + \frac{1}{2^n}},$$

¹ Here and below we assume that $t > C(p + q(n + m + \text{time}(G_{pk})))$ for some absolute constant $C > 0$, so that the attacker can always parse the input, and run G for q times.

$k = |\text{seed}|$, q_G and q_A are the bounds on the number of times G and A query the random oracle, respectively.

Proof. Assume, to the contrary, there exists a pair of algorithms (K, A) running in time t' , such that the triple $(K, f_{\text{seed}} \circ G(\cdot), A(\text{seed}, \cdot))$ is a $(t, q, \delta, (\mathcal{G}_{\text{dist}}, \varepsilon))$ -backdoored pseudorandom generator. I.e.,

$$\text{Adv}_{\text{dist}}^{\text{BPRG}}(K, f_{\text{seed}} \circ G, A) = 2 \left| \Pr[\mathcal{G}_{\text{dist}}^{\text{BPRG}}(K, f_{\text{seed}} \circ G, A) \Rightarrow \text{true}] - \frac{1}{2} \right| > \varepsilon$$

in Game $\mathcal{G}_{\text{dist}}^{\text{BPRG}}(K, f_{\text{seed}} \circ G, A)$ from Fig. 3. Let r_1, \dots, r_q be the outputs of G_{pk} before the immunization, i.e. $s \leftarrow \mathcal{U}, (r_1, \dots, r_q) \leftarrow \text{out}^q(G_{pk}, s)$. The immunization is $f_{\text{seed}}(r_i) = \text{RO}(r_i \parallel \text{seed})$ for $1 \leq i \leq q$.

We define the following three events:

- W_1 : $r_i = r_j$ for $i \neq j$.
- W_2 : G_{pk} queries $(r_i \parallel \text{seed})$ for some $1 \leq i \leq q$.
- W_3 : A queries $(r_i \parallel \text{seed})$ for some $1 \leq i \leq q$.

Note that if none of the events above happened then the two distributions in the distinguishing game corresponding to the challenge bit being 0 or 1, are identical. Now we proceed to bound the probabilities of these three events.

- Since the PRG-security of G is δ , $\Pr[W_1] \leq \frac{q^2}{2^n} + \delta$.
- In the semiprivate model G does not see seed , therefore, the probability that G queries $r_i \parallel \text{seed}$ in one of its queries is the probability that the G guesses seed , and by the union bound this is bounded from above by $\frac{q^G}{2^k}$. Thus, $\Pr[W_2] \leq q^G / 2^k$.
- Now, we look at the probability that A makes a relevant query, given that G did not query $r_i \parallel \text{seed}$ for all i . Assume A predicts r_i for $i \in I \subseteq [q]$. Then there exists $i \in I$ that was predicted first, i.e. when all $f_{\text{seed}}(r_j)$ looked random to A . Then, the probability that A predicts at least one r_i is at most $\sum_{i=1}^q \Pr[A \text{ predicts } r_i \text{ using } q_A \text{ queries given } sk]$. Since A makes at most q_A calls to the random oracle, the latter probability, by the union bound, is bounded by $q_A \sum_{i=1}^q \Pr[A \text{ predicts } r_i \text{ using one query given } sk]$. Now Lemma 4 gives us the following bound:

$$\begin{aligned} \Pr[W_3] &\leq \sum_{i=1}^q \Pr[A \text{ predicts } r_i \text{ using } q_A \text{ queries} | sk] \\ &\leq q_A \sum_{i=1}^q \Pr[A \text{ predicts } r_i \text{ using one query} | sk] \\ &\leq q_A \sum_{i=1}^q \text{Pred}[r_i | sk] \leq q q_A \sqrt{\delta + \frac{1}{2^n}}. \end{aligned}$$

By the claims above,

$$\varepsilon = \Pr[W_1] + \Pr[W_2] + \Pr[W_3] \leq \delta + \frac{q^2}{2^n} + \frac{q_G}{2^k} + qq_A \sqrt{\delta + \frac{1}{2^n}}.$$

□

Positive Result in Standard Model. In this section, we show that replacing the Random Oracle with a UCE function [4] is secure in the standard model. First, we briefly recall Universal Computational Extractor (UCE) defined in [4] by Bellare et al. UCE is a family of security notions for a hash function family.

UCE Security. A notion of UCE security is specified by specifying a class of sources \mathcal{S} . The source is given oracle access to the hash function. UCE security for the class of sources \mathcal{S} states that for any PPT algorithm called the distinguisher D , who receives the key of the hash function and leakage L passed by the source, cannot tell better than random guessing whether H_k was used or a random function. We now give the formal definitions. A source S is a PPT algorithm which is given oracle access to Hash, and outputs a value L called the leakage. For a pair of source S and distinguisher D , define the $\text{UCE}_H^{S,D}$ game as shown in Fig. 11.

Definition 12. A function H is called $\text{UCE}[\mathcal{S}, q_D, \epsilon]$ -secure, if for all sources $S \in \mathcal{S}$, and all polynomial-time algorithms D that make at most q_D queries to H , $\text{Adv}_H^{\text{UCE}}(S, D) := 2\Pr[\text{UCE}_H^{S,D} \Rightarrow \text{true}] - 1 \leq \epsilon$.

For a source S , and a polynomial-time algorithm P called the predictor, define the game Pred_S^P as shown in Fig. 12.

Definition 13. A source S is called (l, ϵ) -statistically unpredictable, denoted by $S \in \mathcal{S}^{\text{sup}}[l, \epsilon]$, if for all computationally unbounded algorithms P that output a list of at most l guesses $\text{Adv}_{S,P}^{\text{Pred}} := \Pr[\text{Pred}_S^P \Rightarrow \text{true}] \leq \epsilon$.

Main $\text{UCE}_H^{S,D}$	$\text{Hash}(x)$
$b \leftarrow \{0, 1\}$	$\text{if } T[x] = \perp \text{ then}$
$k \leftarrow K$	$\text{if } (b = 1) \text{ then}$
$L \leftarrow S^{\text{Hash}}$	$ T[x] \leftarrow H(k, x)$
$b' \leftarrow D(k, L)$	else
return $(b' = b)$	$ T[x] \leftarrow \{0, 1\}^k$
	return $T[x]$

Fig. 11. Game UCE and Hash Oracle

Pred_S^P	$\text{Hash}(x)$
$\text{done} \leftarrow \text{false}$	if $\text{done} \leftarrow \text{false}$ then
$Q \leftarrow \emptyset$	$Q \leftarrow Q \cup \{x\}$
$L \leftarrow S^{\text{Hash}}$	if $T[x] = \perp$ then
$\text{done} \leftarrow \text{true}$	$T[x] \leftarrow \{0, 1\}^k$
$Q' \leftarrow P^{\text{Hash}}(L)$	return $T[x]$
return $(Q \cap Q' \neq \emptyset)$	

Fig. 12. Game Pred and Hash Oracle

Theorem 4. Let $\{G_{pk}: \{0, 1\}^m \rightarrow \{0, 1\}^n \times \{0, 1\}^m | pk \in \{0, 1\}^p\}$ be a (t, q, δ) -pseudorandom generator. Then $f_{\text{seed}}(x) = H_{\text{seed}}(x)$ is a semiprivate $((t, q, \delta), (t, q, \delta + \varepsilon), (\mathcal{G}_{\text{dist}}, \varepsilon'))$ -immunization for G_{pk} , where

$$\varepsilon' = 2\epsilon + \delta + \frac{q^2}{2^n},$$

$$H \in \text{UCE}[\mathcal{S}, q_D, \epsilon], \mathcal{S} = \mathcal{S}^{\text{sup}}[l, \delta + \frac{q^2}{2^n} + ql\sqrt{\delta + \frac{1}{2^n}}].$$

Proof. Given an adversary A playing the distinguishing attack game $\mathcal{G}_{\text{dist}}^{\text{BPRG}}(K, H \circ G, A(\text{seed}))$ we will construct a statistically unpredictable source S and a polynomial-time distinguisher D (see Fig. 13) such that $\text{Adv}_{\text{dist}}^{\text{BPRG}}(K, H \circ G, A(\text{seed})) \leq 2\text{Adv}_H^{\text{UCE}}(S, D) + \delta + \frac{q^2}{2^n}$.

S^{Hash}	$D(d, sk, I, k)$
$(pk, sk) \leftarrow K$	$d' \leftarrow A(sk, I, k)$
$s \leftarrow \{0, 1\}^m$	if $(d = d')$ then
$r_1, r_2, \dots, r_q \leftarrow \text{out}^q(G_{pk}, s)$	return 1
for $1 \leq i \leq q$ do	else
$u_i^0 \leftarrow \text{Hash}(R_i)$	return 0
$u_i^1 \leftarrow \{0, 1\}^n$	
$d \leftarrow \{0, 1\}$	
$I = \{u_1^d, \dots, u_q^d\}$	
return (d, pk, sk, I)	

Fig. 13. Source S and Distinguisher D

Let b be the challenge bit in the UCE game $\text{UCE}_H^{S, D}$. Then,

$$\Pr[\text{UCE}_H^{S, D} \Rightarrow \text{true} | b = 1] = \Pr[\mathcal{G}_{\text{dist}}^{\text{BPRG}}(K, H \circ G, A(\text{seed})) \Rightarrow \text{true}],$$

$$\Pr[\text{UCE}_H^{S, D} \Rightarrow \text{true} | b = 0] = 1 - \Pr[\mathcal{G}_{\text{dist}}^{\text{BPRG}}(K, \text{RO} \circ G, A(\text{seed})) \Rightarrow \text{true}],$$

where in the RO immunization game, A has to distinguish uniformly random outputs from RO applied to the outputs of G. If r 's are distinct, then these two distributions are identical. From the PRG security, the probability of the event $r_i = r_j$ for $i \neq j$ is less than $\delta + \frac{q^2}{2^n}$. Therefore,

$$\Pr[\text{UCE}_H^{S,D} \Rightarrow \text{true} | b = 0] \geq \frac{1}{2} - \frac{1}{2}(\delta + \frac{q^2}{2^n})$$

Summing yields,

$$\begin{aligned} \text{Adv}_H^{\text{UCE}}(S, D) &= \frac{1}{2} \text{Adv}_{\text{dist}}^{\text{BPRG}}(K, H \circ G, A) - \frac{1}{2}\delta - \frac{1}{2} \cdot \frac{q^2}{2^n}, \\ \text{Adv}_{\text{dist}}^{\text{BPRG}}(K, H \circ G, A) &\leq 2\text{Adv}_H^{\text{UCE}}(S, D) + \delta + \frac{q^2}{2^n}. \end{aligned}$$

Now we argue that S is statistically unpredictable; that is, it is hard to guess the source's Hash queries even given the leakage, in the random case of the UCE game. Consider an arbitrary predictor P , and the advantage of P in the game Pred_S^P . If all R_i are distinct (which happens with probability $1 - \delta - \frac{q^2}{2^n}$), the probability that P guesses at least one of r 's given the leakage is at most $q\text{Pred}(R|sk)$. Now, since P outputs a list of length l , by Lemma 4,

$$\text{Adv}_{S,P}^{\text{Pred}} \leq \delta + \frac{q^2}{2^n} + ql\sqrt{\delta + \frac{1}{2^n}}.$$

□

5.3 Private Immunization Model

We now study the strongest model of immunization which is the private model, where seed is secret from both the PRG and the attacker. We show that a PRF is an immunization function in this model. But if users had access to a backdoor-less PRF, then instead of using it to immunize a backdoored PRG, they could use the PRF itself for pseudorandomness. In this section, we explore using functions weaker than PRF as immunization functions, and show that some natural functions are not secure immunizations.

PRF Immunization

Lemma 5. *Let $\{G_{pk}: \{0,1\}^m \rightarrow \{0,1\}^n \times \{0,1\}^m | pk \in \{0,1\}^p\}$ be a (t, q, δ) -pseudorandom generator, let also $\{f_{\text{seed}}: \{0,1\}^n \rightarrow \{0,1\}^k | \text{seed} \in \{0,1\}^l\}$ be a (t, q, ε) -pseudorandom function. Then f_{seed} is a private $((t, q, \delta), (t, q, \delta + \varepsilon), (\mathcal{G}_{\text{dist}}, \varepsilon'))$ -immunization for G_{pk} , where*

$$\varepsilon' = \varepsilon + \delta + \frac{q^2}{2^n}.$$

Proof. From the definition of PRF, no distinguisher D running in time t given q outputs of F_{seed} can distinguish the output from uniformly random with advantage greater than ε . By PRG security of G_{pk} , $\mathbf{CD}_t((pk, \text{out}^q(G_{pk}, \mathcal{U})), \mathcal{U}) \leq \delta$. Therefore, $\{f_{\text{seed}} \circ G_{pk}(\cdot) | (\text{seed}, pk) \in \{0,1\}^\ell \times \{0,1\}^p\}$ is a $(t, q, \delta + \varepsilon)$ -pseudorandom generator. Similar to the proof of Theorem 3, $\text{Adv}_{\text{dist}}^{\text{BPRG}}(K, f_{\text{seed}} \circ G(\cdot), A(\cdot)) \leq \text{Adv}_f^{\text{PRF}} + \Pr[\exists i, j : r_i = r_j | (r_1, \dots, r_q) \leftarrow \text{out}^q(G_{pk}, \mathcal{U})] \leq \varepsilon + \delta + \frac{q^2}{2^n}$. \square

Attack against XOR. One of natural candidates for the immunization function in the private randomness model is the function XOR with a random string as private randomness. In this section we show an attack against $f_{\text{seed}}(x) = \text{seed} \oplus x$, where seed is private randomness of immunization function f . The backdoored PRG works as follows: it outputs strings such that the XOR of two consecutive outputs leaks one bit of s_1 where s_1 is a part of the seed of length n , such that the bias introduced is negligible. After $(n+1)$ outputs A can recover all of s_1 , and can predict future outputs.

Lemma 6. *Let $(K, \text{Enc}_{pk}, \text{Dec}_{sk})$ be a $(t, q, \delta) - \text{IND\$-CPA}$ public key encryption scheme, G be a (t, q, δ) -pseudorandom generator. Then for (K', G', A') defined in Fig. 14 and $f_{\text{seed}}(x) = \text{seed} \oplus x$, $(K', f_{\text{circ}} \circ G'(\cdot), A'(\cdot))$ is a $(t, q - n \frac{\log n}{1-\delta}, 2\delta, (\mathcal{G}_{\text{next}}, \varepsilon))$ -backdoored pseudorandom generator, where $\varepsilon = 1 - n \cdot \exp\left(\frac{-\ln^2 n}{3(1-\delta)}\right)$.*

Proof. From the Chernoff bound:

$$\text{Adv}_{\text{next}}^{\text{BPRG}}(K', G', A') \geq 1 - n \cdot \Pr[\text{count}_2 \geq \frac{\ln n}{1-\delta}] \geq 1 - n \cdot \exp\left(\frac{-\ln^2 n}{3(1-\delta)}\right).$$

From pseudorandomness of G 's outputs $\mathbf{CD}_t(\text{out}^q(G_{pk}, \mathcal{U}), \mathcal{U}) \leq \delta$, and $\mathbf{CD}_t((pk, s_1, \text{Enc}(s_1)), (pk, s_1, \mathcal{U})) \leq \delta$ due to IND\$-CPA security. Thus, $\mathbf{CD}_t((pk, \text{out}^q(G'_{pk}, \mathcal{U})), \mathcal{U}) \leq 2\delta$. \square

Extensions of XOR-attack. The previous attack can be extended in two ways. First, the PRG can be modified so that one does not need to see $q > n$ outputs to guess the next one, with high probability it is enough to see just three random outputs. Although this kind of attack is weaker than the definition of rseek-attack, it is much stronger than next-attack. Second, using homomorphic encryption, the previous attack can be extended to some other natural immunization functions. Here we show an example where the multiplication with a private random string is not a private immunization. Let $(K, \text{Enc}_{pk}, \text{Dec}_{sk})$ be a homomorphic $(t, q, \delta) - \text{IND\$-CPA}$ encryption scheme. For simplicity we assume that $\text{Enc}_{pk} : \mathbb{Z}_b \rightarrow \mathbb{Z}_n$, $\text{Dec}_{sk} : \mathbb{Z}_n \rightarrow \mathbb{Z}_b$, and $\text{Enc}_{pk}(m_1) \cdot \text{Enc}_{pk}(m_2) = \text{Enc}((m_1 + m_2) \bmod b)$, and the immunization function $f_{\text{seed}}(r) = (\text{seed} \cdot r) \bmod n$ (e.g., one can think of Benaloh cryptosystem [6]).

By 3rseek we mean the rseek-game where the adversary gets to see 3 outputs rather than just one.

$\underline{K'}$	$G'(pk, (s_0, s_1, c, r_{\text{prev}}, \text{count}))$	$A'(sk, f_{\text{seed}}(r_1), \dots, f_{\text{seed}}(r_q))$
$(pk, sk) \leftarrow K$	$s'_0 \leftarrow s_0$	$\text{for } 1 \leq i \leq n \text{ do}$
return (pk, sk)	$s'_1 \leftarrow s_1$	$ \quad c_{(i)} \leftarrow (f_{\text{seed}}(r_i))^{(1)} \oplus$
	if $\text{count} = 1$ then	$c = c_{(1)}c_{(2)} \dots c_{(n)}$
	$ \quad (\alpha, s_0) \leftarrow G(s_0)$	$s_1 \leftarrow \text{Dec}_{sk}(c)$
	$ \quad c \leftarrow \text{Enc}_{pk}(s_1; \alpha)$	$r'_{n+2} \leftarrow G(s_1)$
	$ \quad n \leftarrow c $	$\text{seed}' \leftarrow r'_{n+2} \oplus f(\text{seed}, r'_{n+2})$
	$ \quad (r', s'_0) \leftarrow G(s_0)$	for $n + 1 < j \leq q + 1$ do
	if $1 < \text{count} \leq n + 1$ then	$ \quad (r'_j, s_1) \leftarrow G(s_1)$
	$ \quad \text{count}_2 \leftarrow 0$	return $r'_{q+1} \oplus \text{seed}'$
	repeat	
	$ \quad (r', s'_0) \leftarrow G(s'_0)$	
	until $((r' \oplus r_{\text{prev}})_{(1)} = c_{(i)}) \vee$	
	$(\text{count}_2 > \frac{\ln n}{1-\delta});$	
	if $\text{count} > (n + 1)$ then	
	$ \quad (r', s'_1) \leftarrow G(s_1)$	
	$r_{\text{prev}} = r'$	
	$\text{count} \leftarrow \text{count} + 1$	
	return	
	$(r', (s'_0, s'_1, c, r_{\text{prev}}, \text{count}))$	

Fig. 14. Predicting backdoored PRG — Private immunization with $f_{\text{seed}}(x) = \text{seed} \oplus x$

$\underline{K'}$	$G'(pk, s_0, s_1, \text{count})$	$A'(sk, f_{\text{seed}}(r_a), f_{\text{seed}}(r_b), f_{\text{seed}}(r_c), d)$
$(pk, sk) \leftarrow K$	$\alpha \leftarrow F_{s_1}(\text{count})$	$e \leftarrow (f_{\text{seed}}(r_a))/f_{\text{seed}}(r_b)$
return (pk, sk)	if $(\text{lsb}_2(\alpha) = 00)$ then	$s'_0 \leftarrow \text{Dec}_{sk}(e)$
	$ \quad r' \leftarrow \text{Enc}_{pk}(0; \alpha^{\gg 2})$	if $s'_0 \neq \perp$ then
	else if $(\text{lsb}_2(\alpha) = 10)$ then	$ \quad r'_c \leftarrow F_{s'_0}(c)$
	$ \quad r' \leftarrow 1/(\text{Enc}_{pk}(s_0; \alpha^{\gg 2}))$	$ \quad \text{seed}' \leftarrow f_{\text{seed}}(r_c)/r'_c$
	else	$ \quad r'_d \leftarrow F_{s'_0}(d) \cdot \text{seed}'$
	$ \quad r' \leftarrow F_{s_0}(\text{count})$	return r'_d
	return $(r', (s_0, s_1, \text{count} + 1))$	return 0

Fig. 15. Predicting Backdoored PRG — Private Immunization

Lemma 7. Let $(K, \text{Enc}_{pk}, \text{Dec}_{sk})$ be a (t, q, δ) -IND\$-CPA public key encryption scheme which is multiplicatively homomorphic as above, F_{sk} be a (t, q, δ) -pseudorandom function for $q \geq 4$. Then for (K', G', A') defined in Fig. 15 and $f_{\text{seed}}(x) = \text{seed} \cdot x$, $(K', f_{\text{seed}} \circ G'(\cdot), A'(\cdot))$ is a $(t, q, 3\delta, (\mathcal{G}_{3\text{seek}}, \frac{1}{64} - \delta))$ -backdoored pseudorandom generator.

Proof. From pseudorandomness of F 's outputs

$$\mathbf{CD}_t((F_{s_0}(1), \dots, F_{s_0}(q)), \mathcal{U}) \leq \delta, \mathbf{CD}_t((F_{s_1}(1), \dots, F_{s_1}(q)), \mathcal{U}) \leq \delta.$$

Then $\mathbf{CD}_t((pk, s_0, \text{Enc}(s_0; \alpha^{\gg 2})), (pk, s_0, \mathcal{U})) \leq 2\delta$ due to IND\$-CPA security. Thus,

$$\mathbf{CD}_t((pk, \text{out}^q(G'_{pk}), \mathcal{U})), \mathcal{U}) \leq 3\delta.$$

$$\begin{aligned}
\text{Adv}_{3rseek}^{\text{BPRG}}(K', f_{\text{seed}} \circ G', A') &= \Pr[r_d = r'_d] \geq \\
\Pr[\text{lsb}(F_{s_1}(d)) = 1 \wedge \text{seed}' = \text{seed} \wedge s'_0 = s_0] &\geq \\
\Pr[\text{lsb}(F_{s_1}(d)) = 1 \wedge r'_c = r_c \wedge s'_0 = s_0] &\geq \\
\Pr[\text{lsb}(F_{s_1}(d)) = 1 \wedge \text{lsb}(F_{s_1}(c)) = 1 \wedge s'_0 = s_0] &\geq \\
\Pr[\text{lsb}(F_{s_1}(d)) = 1 \wedge \text{lsb}(F_{s_1}(c)) = 1 \wedge r_a = \text{Enc}_{pk}(0) \wedge r_b = 1 / (\text{Enc}_{pk}(s_0))] &\geq \\
\Pr[\text{lsb}(F_{s_1}(d)) = 1 \wedge \text{lsb}(F_{s_1}(c)) = 1 \wedge \text{lsb}_2(F_{s_1}(a)) = 00 \wedge \text{lsb}_2(F_{s_1}(b)) = 10] &\geq \\
&\quad \frac{1}{64} - \delta
\end{aligned}$$

for $q \geq 4$.

□

References

- Albertini, A., Aumasson, J.P., Eichlseder, M., Mendel, F., Schläffer, M.: Malicious hashing: Eve's variant of SHA-1. Cryptology ePrint Archive, Report 2014/694 (2014). <http://eprint.iacr.org/>
- Aranha, D.F., Fouque, P.A., Qian, C., Tibouchi, M., Zapalowicz, J.C.: Binary elliptic squared. Cryptology ePrint Archive, Report 2014/486 (2014). <http://eprint.iacr.org/>
- Backes, M., Cachin, C.: Public-key steganography with active attacks. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 210–226. Springer, Heidelberg (2005)
- Bellare, M., Hoang, V.T., Keelveedhi, S.: Instantiating random oracles via UCes. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 398–415. Springer, Heidelberg (2013)
- Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against aass surveillance. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 1–19. Springer, Heidelberg (2014)
- Benaloh, J.: Dense probabilistic encryption. In: Proceedings of the Workshop on Selected Areas of Cryptography, pp. 120–128 (1994)
- Bendel, M.: Hackers describe PS3 security as epic fail, gain unrestricted access. <http://www.exophase.com/20540/hackers-describe-ps3-security-as-epic-fail-gain-unrestricted-access/>
- Bernstein, D.J., Chang, Y.-A., Cheng, C.-M., Chou, L.-P., Heninger, N., Lange, T., van Someren, N.: Factoring RSA keys from certified smart cards: coppersmith in the wild. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 341–360. Springer, Heidelberg (2013)
- 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)
- Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM Journal on Computing **15**(2), 364–383 (1986)
- Brown, D., Vanstone, S.: Elliptic curve random number generation (2007). <http://www.google.com/patents/US20070189527>

12. Cachin, C.: An information-theoretic model for steganography. In: Aucsmith, D. (ed.) IH 1998. LNCS, vol. 1525, pp. 306–318. Springer, Heidelberg (1998)
13. Checkoway, S., Fredrikson, M., Niederhagen, R., Green, M., Lange, T., Ristenpart, T., Bernstein, D.J., Maskiewicz, J., Shacham, H.: On the practical exploitability of Dual EC DRBG in TLS implementations (2014)
14. Everspaugh, A., Zhai, Y., Jellinek, R., Ristenpart, T., Swift, M.: Not-so-random numbers in virtualized linux and the Whirlwind RNG (2014)
15. Goh, E.-J., Boneh, D., Pinkas, B., Golle, P.: The design and implementation of protocol-based hidden key recovery. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 165–179. Springer, Heidelberg (2003)
16. Goldberg, I., Wagner, D.: Randomness and the Netscape browser. Dr Dobb's Journal pp. 66–71 (1996)
17. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: Detection of widespread weak keys in network devices. In: USENIX Security, pp. 205–220. USENIX (2012)
18. Holenstein, T.: Key agreement from weak bit agreement. In: Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, pp. 664–673. ACM (2005)
19. Hopper, N., von Ahn, L., Langford, J.: Provably secure steganography. IEEE Transactions on Computers **58**(5), 662–676 (2009)
20. Juels, A., Guajardo, J.: RSA key generation with verifiable randomness. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 357–374. Springer, Heidelberg (2002)
21. Möller, B.: A public-key encryption scheme with pseudo-random ciphertexts. In: Samarati, P., Ryan, P.Y.A., Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 335–351. Springer, Heidelberg (2004)
22. Mowery, K., Wei, M., Kohlbrenner, D., Shacham, H., Swanson, S.: Welcome to the Entropics: Boot-time entropy in embedded devices, pp. 589–603. IEEE (2013)
23. National Institute of Standards and Technology: Special Publication 800–90: Recommendation for random number generation using deterministic random bit generators (2012), <http://csrc.nist.gov/publications/PubsSPs.html#800-90A>, (first version June 2006, second version March 2007)
24. Ristenpart, T., Yilek, S.: When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In: NDSS (2010)
25. Schoenmakers, B., Sidorenko, A.: Cryptanalysis of the dual elliptic curve pseudo-random generator. IACR Cryptology ePrint Archive **2006**, 190 (2006)
26. Shoup, V.: A proposal for an iso standard for public key encryption (version 2.1). IACR E-Print Archive 112 (2001)
27. Shumow, D., Ferguson, N.: On the possibility of a back door in the NIST SP800-90 Dual Ec Prng. In: Proc. Crypto 2007 (2007)
28. Simmons, G.J.: The prisoners' problem and the subliminal channel. In: Advances in Cryptology. pp. 51–67. Springer (1984)
29. Tibouchi, M.: Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. Cryptology ePrint Archive, Report 2014/043 (2014). <http://eprint.iacr.org/>
30. Vazirani, U.V., Vazirani, V.V.: Trapdoor pseudo-random number generators, with applications to protocol design. FOCS **83**, 23–30 (1983)
31. Vazirani, U.V., Vazirani, V.V.: Efficient and secure pseudo-random number generation. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 193–202. Springer, Heidelberg (1985)

32. von Ahn, L., Hopper, N.J.: Public-key steganography. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 323–341. Springer, Heidelberg (2004)
33. Yilek, S., Rescorla, E., Shacham, H., Enright, B., Savage, S.: When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In: SIGCOMM Conference on Internet Measurement, pp. 15–27. ACM (2009)
34. Young, A., Yung, M.: The dark side of “black-box” cryptography, or: should we trust capstone? In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 89–103. Springer, Heidelberg (1996)
35. Young, A., Yung, M.: Kleptography: using cryptography against cryptography. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 62–74. Springer, Heidelberg (1997)
36. Young, A., Yung, M.: Kleptography from standard assumptions and applications. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 271–290. Springer, Heidelberg (2010)

Number Field Sieve

Improving NFS for the Discrete Logarithm Problem in Non-prime Finite Fields

Razvan Barbulescu^{1,2,3(✉)}, Pierrick Gaudry^{1,3,4}, Aurore Guillevic^{1,2},
and François Morain^{1,2,3}

¹ Institut National de Recherche en Informatique et en Automatique (INRIA),
Paris, France

razvan.barbaud@inria.fr

² École Polytechnique/LIX, Palaiseau, France

{guillevic,morain}@lix.polytechnique.fr

³ Centre National de la Recherche Scientifique (CNRS), Paris, France

⁴ Université de Lorraine, Nancy, France

pierrick.gaudry@loria.fr

Abstract. The aim of this work is to investigate the hardness of the discrete logarithm problem in fields $GF(p^n)$ where n is a small integer greater than 1. Though less studied than the small characteristic case or the prime field case, the difficulty of this problem is at the heart of security evaluations for torus-based and pairing-based cryptography. The best known method for solving this problem is the Number Field Sieve (NFS). A key ingredient in this algorithm is the ability to find good polynomials that define the extension fields used in NFS. We design two new methods for this task, modifying the asymptotic complexity and paving the way for record-breaking computations. We exemplify these results with the computation of discrete logarithms over a field $GF(p^2)$ whose cardinality is 180 digits (595 bits) long.

1 Introduction

The security of cryptographic protocols relies on hard problems like integer factorization or discrete logarithm (DLP) computations in a finite group. The difficulty of the latter depends on the chosen group. While no subexponential methods for DLP instances are known for some groups (including elliptic curves), finite fields are vulnerable to variants of the Number Field Sieve (NFS) algorithm.

Getting more insight about the theoretical and the practical behaviour of NFS for non-prime fields is important in cryptography. Indeed, although cryptosystems based on discrete logarithms in non-prime finite fields are not as widely deployed as for prime fields, they can be found in two areas: torus-based and pairing-based cryptography.

Torus-based cryptography, and in particular its most popular avatars LUC [32], XTR [21] and CEILIDH [29], provides an efficient way to build a cryptosystem working in a subgroup of the multiplicative group of a finite field \mathbb{F}_{p^n} where

$n > 1$ is a small integer. The size of the considered prime-order subgroup must be large enough to resist Pollard's rho attacks, and p^n must be large enough to resist NFS attacks.

In pairing-based cryptography, the security relies on the difficulty of the discrete logarithm problem in elliptic curves, and in finite fields of the form \mathbb{F}_{q^n} , where n is small (mostly $n \leq 20$). The table [12, Tab. 1.1] lists the available choices of elliptic curve and finite field sizes to balance the security on both sides. Due to recent progress in attacks on discrete logarithms in finite fields of small characteristic, it is also common to assume that q is prime, so that NFS is also to be considered as an attack to be resisted.

Expressing the complexity of subexponential methods is done using the L -function. If $\alpha \in [0, 1]$ and $c > 0$ are two constants, we set

$$L_Q(\alpha, c) = \exp((c + o(1))(\log Q)^\alpha (\log \log Q)^{1-\alpha}),$$

and simply write $L_Q(\alpha)$ if the constant c is not made explicit.

It was proven in [16] that the complexity of DLP in the medium and large characteristic case is $L_Q(1/3, c)$. This ended proving that the complexity of DLP for every finite field of cardinality Q is $L_Q(1/3, c)$. The constant c depends on the size of the characteristic p with respect to Q . It is customary to write $p = L_Q(\alpha, c)$ and to classify the different cases as follows: p is said to be *small* if $\alpha < 1/3$, *medium* if $1/3 < \alpha < 2/3$ and *large* if $2/3 < \alpha$. In this article, we target non-small characteristic finite fields, for which the state-of-art algorithm is the Number Field Sieve (NFS) and where the quasi polynomial time algorithm [2] does not apply. Schirokauer [30] studied the family of fields \mathbb{F}_{p^n} for which n is constant when p goes to infinity, and obtained the same complexity as in the prime case $L_Q(1/3, \sqrt[3]{64/9})$. The variants of the algorithm by Joux, Lercier, Smart and Vercauteren [16] have complexity $L_Q(1/3, \sqrt[3]{64/9})$ for fields of large characteristic, and $L_Q(1/3, \sqrt[3]{128/9})$ in medium characteristic. A Coppersmith-like variant [3] has a complexity of $L_Q(1/3, \sqrt[3]{(92 + 26\sqrt{13})/27})$ in large characteristic, and $L_Q(1/3, \sqrt[3]{2^{13}/3^6})$ in medium characteristic. The situation is more complex in the boundary case, i.e. when $p = L_Q(2/3)$, having a complexity $L_Q(1/3, c)$ with c varying between $16/9$ and $\sqrt[3]{2^{13}/3^6}$ [3]. Finally, if the characteristic p has a special form, i.e. can be written as $p = P(m)$, for an integer $m \approx p^{1/\deg P}$ and a polynomial $P \in \mathbb{Z}[x]$ of small degree and with small coefficients, then a faster variant of NFS is available [18].

In practice, the boundary between the medium and large characteristic cases is determined by direct timing of each variant of NFS, for each pair $(\lfloor \log p \rfloor, n)$. A series of record computations were realized using the medium characteristic variant [17, Table 8.], but to our knowledge no computations have been done in non-prime fields using the large characteristic variant of NFS.

In this article, we propose two new methods to select polynomials for NFS in the context of discrete logarithms in non-prime finite fields: the Generalized Joux–Lercier (GJL) method and the Conjugation (Conj) method.

We prove the following result in Section 4.

Theorem 1. Let \mathbb{F}_Q , with $Q = p^n$ be a finite field of characteristic p . Assuming the usual heuristics on smoothness of algebraic numbers, made in the analysis of the Number Field Sieve algorithm, we get the following time complexities for computing a discrete logarithm in \mathbb{F}_Q , depending on the polynomial selection method and the size of p compared to Q :

1. (Large prime case). Assuming $p \geq L_Q(2/3, \sqrt[3]{8/3})$, NFS with the Generalized Joux–Lercier method has complexity

$$L_Q\left(1/3, \sqrt[3]{64/9}\right).$$

2. (Medium prime case). Assuming $p = L_Q(\alpha)$ for $1/3 < \alpha < 2/3$, NFS with the Conjugation method has complexity

$$L_Q\left(1/3, \sqrt[3]{96/9}\right).$$

3. (Boundary case). Assuming $p = L_Q(2/3, 12^{1/3})^{1+o(1)}$, NFS with the Conjugation method has complexity

$$L_Q\left(1/3, \sqrt[3]{48/9}\right).$$

This improves on the previously known complexities in the medium characteristic case, and in the boundary case where p is around $L_Q(2/3)$.

When the characteristic is large, we improved on the known method of the polynomial selection (Proposition 5), but our gain is hidden in the $1 + o(1)$ exponent of the complexity formula. It led us to do a more precise analysis for sizes around the limit of what seems feasible with current and near future technology, and for extension degrees between 2 and 6. This suggests that our polynomial selection methods behave better than expected for small extension degrees.

In order to illustrate this, we implemented our methods and ran a discrete logarithm computation in a finite field of the form \mathbb{F}_{p^2} , where p has 90 decimal digits. With the Conjugation method, we were able to perform this in a time that is smaller than what is needed to factor an integer of 180 decimal digits.

Outline. The paper is organized as follows. In Section 2 we make a short presentation of the number field sieve. In Section 3 we present two new methods of polynomial selection for NFS. We measure their asymptotic complexity, thus proving Theorem 1, and derive non-asymptotic cost estimates for small degree extensions in Section 4. After discussing some further improvements in Section 5, we detail a record computation in \mathbb{F}_{p^2} obtained with our algorithm in Section 6.

2 Sketch of the Number Field Sieve

Let us sketch the variant of NFS, the state-of-art algorithm used to compute discrete logarithms in any finite field \mathbb{F}_{p^n} with non-small characteristic.

In the first stage, called *polynomial selection*, two polynomials f, g in $\mathbb{Z}[x]$ are constructed (we assume that $\deg f \geq \deg g$), such that their reductions modulo p have a common irreducible factor $\varphi \in \mathbb{F}_p[x]$ which is irreducible of degree n . This polynomial φ defines \mathbb{F}_{p^n} over \mathbb{F}_p , and does not impact the complexity of NFS. We will use it in the final part of the algorithm, the individual logarithm stage (see below), to embed the input element, whose discrete logarithm is requested, into our representation $\mathbb{F}_{p^n} = \mathbb{F}_p[x]/\langle \varphi \rangle$.

Essentially, we impose no additional condition on f and g . During this presentation we will consider the number fields of f and g , meaning that the two polynomials are irreducible, but everything works the same if we replace them by their irreducible factors over \mathbb{Z} which are divisible by φ modulo p . Far more important is the fact that we do not make the classical assumption that f and g are monic. Indeed, the NFS algorithm can be modified to work with non-monic polynomials by adding factors of the leading coefficient to the factor base and by implementing carefully the decomposition of algebraic numbers into prime ideals. This was known in the folklore of NFS for a long time now and was also written more or less explicitly in the literature of NFS [16, 20] where non-monic polynomials are used. A recent description of this technicalities is made in Section 6.5 of [4]. CADO-NFS accepts non-monic polynomials as an input, which allowed us to do experiments with our polynomials.

Let α and β be algebraic numbers such that $f(\alpha) = 0$ and $g(\beta) = 0$ and let m be a root of φ in \mathbb{F}_{p^n} , allowing us to write $\mathbb{F}_{p^n} = \mathbb{F}_p(m)$. Let K_f and K_g be the number fields associated with f and g respectively, and \mathcal{O}_f and \mathcal{O}_g their rings of integers. In the algorithm, we consider elements of $\mathbb{Z}(\alpha)$ and $\mathbb{Z}(\beta)$ as polynomials in α and β respectively, which are in general not integers. However, the only denominators that can occur are divisors of the leading coefficients of the polynomials f and g that we denote respectively by $l(f)$ and $l(g)$.

For the second stage of NFS, called *relation collection* or *sieving*, a smoothness bound B is chosen and we consider the two factor bases

$$\mathcal{F}_f = \left\{ \begin{array}{l} \text{prime ideals } \mathfrak{q} \text{ in } \mathcal{O}_f \text{ of norm less than } B \\ \text{or above prime factors of } l(f) \end{array} \right\},$$

$$\mathcal{F}_g = \left\{ \begin{array}{l} \text{prime ideals } \mathfrak{q} \text{ in } \mathcal{O}_g \text{ of norm less than } B \\ \text{or above prime factors of } l(g) \end{array} \right\}.$$

An integer is B -smooth if all its prime factors are less than B . For any polynomial $\phi(x) \in \mathbb{Z}[x]$, the algebraic number $\phi(\alpha)$ (resp. $\phi(\beta)$) in K_f (resp. K_g) is B -smooth if $\text{Res}(f, \phi)$ (resp. $\text{Res}(g, \phi)$) is the product of a B -smooth integer and of a product of factors of $l(f)$ (resp. $l(g)$). In that case, due to the equation

$$N(\phi(\alpha)) = \pm l(f)^{-\deg \phi} \text{Res}(f, \phi),$$

the fractional ideal $\phi(\alpha)\mathcal{O}_f$ decomposes into a product of ideals of \mathcal{F}_f , with positive or negative exponents.

In the sieve stage, one collects $\#\mathcal{F}_f + \#\mathcal{F}_g$ polynomials $\phi(x) \in \mathbb{Z}[x]$ with coprime coefficients and degree at most $t - 1$, for a parameter $t \geq 2$ to be

chosen, such that $\text{Res}(f, \phi)$ and $\text{Res}(g, \phi)$ are B -smooth. Since both $\phi(\alpha)$ and $\phi(\beta)$ are B -smooth, we get *relations* of the form:

$$\begin{cases} \phi(\alpha)\mathcal{O}_f = \prod_{\mathfrak{q} \in \mathcal{F}_f} \mathfrak{q}^{\text{val}_{\mathfrak{q}}(\phi(\alpha))} \\ \phi(\beta)\mathcal{O}_g = \prod_{\mathfrak{r} \in \mathcal{F}_g} \mathfrak{r}^{\text{val}_{\mathfrak{r}}(\phi(\beta))}. \end{cases}$$

Each relation allows us to write a linear equation among the (virtual) logarithms of the ideals in the factor base. In this article we hush up the technical details related to virtual logarithms and Schirokauer maps, but the reader can find a detailed description in [5, 16, 31].

The norm of $\phi(\alpha)$ (resp. of $\phi(\beta)$) is the product of the norms of the ideals in the right hand side and will be bounded by the size of the finite field $(p^n)^{\max(\deg f, \deg g)}$ (this is a very crude estimate; refining it is the heart of the complexity analysis of Section 4); therefore the number of ideals involved in a relation is less than $\log_2(p^n)^{O(1)}$. One can also remark that the ideals that can occur in a relation have degrees that are at most equal to the degree of ϕ , that is $t - 1$. Therefore, it makes sense to include in \mathcal{F}_f and \mathcal{F}_g only the ideals of degree at most $t - 1$ (for a theoretical analysis of NFS one can maintain the variant without restrictions on the degree of the ideals in the factor base).

In order to estimate the probability that a random polynomial ϕ with given degree and size of coefficients gives a relation, we make the common heuristic that the integer $\text{Res}(\phi, f) \cdot \text{Res}(\phi, g)$ has the same probability of B -smoothness as a random integer of the same size. Therefore, reducing the expected size of this product of norms is the main criterion when selecting the polynomials f and g .

In the *linear algebra* stage, we consider the homogeneous linear system obtained after the sieve. We make the usual heuristic that this system has a space of solutions of dimension one. Since the system is sparse (at most $(\log_2(p^n))^{O(1)}$ non-zero entries per row), an iterative algorithm like (block) Wiedemann [10, 33] is used to compute a non-zero solution in quasi-quadratic time. This gives the (virtual) logarithms of all the factor base elements.

In principle, the coefficient ring of the matrix is $\mathbb{Z}/(p^n - 1)\mathbb{Z}$, but it is enough to solve it modulo each prime divisor ℓ of $p^n - 1$ and then to recombine the results using the Pohlig-Hellman algorithm [27]. Since one can use Pollard's method [28] for small primes ℓ , we can suppose that ℓ is larger than $L_{p^n}(1/3)$. Since ℓ is large, we may assume that ℓ is coprime to $\text{Disc}(f)$, $\text{Disc}(g)$, the class numbers of K_f and K_g , and the orders of the roots of unity in K_f and K_g . These assumptions greatly simplify the theory, but again, we do not elaborate on the mathematical aspects of the algorithm since the improvements that we discuss in this article do not affect them.

In the last stage of the algorithm, called *individual logarithm*, the discrete logarithm of any element $z = \sum_{i=0}^{n-1} z_i m^i$ of \mathbb{F}_{p^n} in the finite field is computed. For this, we associate z with the algebraic number $\bar{z} = \sum_{i=0}^{n-1} z_i \alpha^i$ in K_f and check whether the corresponding principal ideal factors into prime ideals of norms bounded by a second bound B' larger than B . We also ask the prime ideals to

be of degree at most $t - 1$. If \bar{z} does not satisfy these smoothness assumptions, then we replace z by z^e for a randomly chosen integer e and try again. This allows us to obtain a linear equation similar to those of the linear system, in which one of the unknowns is $\log z$. The second step of the individual logarithm stage consists in obtaining relations between a prime ideal and prime ideals of smaller norm, until all the ideals involved are in \mathcal{F}_f or \mathcal{F}_g . This allows us to backtrack and obtain $\log z$.

3 New Methods for Selecting Polynomials

In this section, we propose two new methods to select the polynomials f and g , in the case of finite fields that are low degree extensions of prime fields. For any polynomial g , we denote by $\|g\|_\infty$ the maximum absolute value of its coefficients. The first method is an extension to non-prime fields of the method used by Joux and Lercier [15] for \mathbb{F}_p . In the second one, we use rational reconstruction and the existence of some square roots in \mathbb{F}_p .

All the constructions that follow use either LLL reduction [22] or simple rational reconstruction (also known as the continued fraction method), see for example [8]. It allows us to write any residue a modulo a prime p as

$$a \equiv u/v \pmod{p}$$

with $u, v \leq c\sqrt{p}$ for some constant c . If one computes the rational reconstruction using LLL in dimension two, then one can show that it always succeeds if c is a large enough explicit constant. In practice we might want two different rational reconstructions $a \equiv u_1/v_1 \equiv u_2/v_2 \pmod{p}$. In this case we cannot make any proof on the size of u_2 and v_2 , but they can be small.

For ease of reading, f is supposed to have degree greater than or equal to that of g . Although the polynomial $\varphi(x)$ that defines \mathbb{F}_{p^n} as $\mathbb{F}_p[X]/\langle \varphi(x) \rangle$ does not occur explicitly in the computations, we sometimes give it along with the pair (f, g) .

3.1 State of the Art

Joux, Lercier, Smart and Vercauteren [16] introduced two methods of polynomial selection in non-prime fields which are the only option for their respective range of application: medium characteristic and, respectively, large characteristic finite fields.

JLSV₁. In Algorithm 1 we recall the method introduced in [16, §2.3]. This method is best suited to the medium characteristic case. It produces two polynomials f and g of the same degree n , which have coefficients of size $O(\sqrt{p})$ each.

Example 1. Take $p = 1000001447$, $n = 4$, and $a = 44723 \geq \lceil \sqrt{p} \rceil$. One has $f = (x^4 - 6x^2 + 1) - 44723(x^3 - x)$ and $g = 22360(x^4 - 6x^2 + 1) - 4833(x^3 - x)$ with $u/v = 4833/22360$ a rational reconstruction of a modulo p .

Algorithm 1. Polynomial selection with the first method in [16] (JLSV₁)

Input: p prime and n integer
Output: f, g, φ with $f, g \in \mathbb{Z}[x]$ irreducible and $\varphi = \gcd(f \bmod p, g \bmod p)$ in $\mathbb{F}_p[x]$ irreducible of degree n

- 1 Select $f_1(x), f_0(x)$, two polynomials with small integer coefficients,
 $\deg f_1 < \deg f_0 = n$;
- 2 **repeat**
- 3 | choose $a \geq \lceil \sqrt{p} \rceil$;
- 4 **until** $f = f_0 + af_1$ is irreducible in $\mathbb{F}_p[x]$;
- 5 $(u, v) \leftarrow$ a rational reconstruction of a modulo p ;
- 6 $g \leftarrow vf_0 + uf_1$;
- 7 **return** $(f, g, \varphi = g \bmod p)$

JLSV₂. In Algorithm 2 we reproduce the method described in [16, §3.2]. We denote by $\text{LLL}(M)$ the matrix obtained by applying the LLL algorithm to the rows of a matrix M with integer coefficients.

Algorithm 2. Polynomial selection with the second method in [16] (JLSV₂)

Input: p prime, n integer and $D \geq n$ integer
Output: f, g, φ with $f, g \in \mathbb{Z}[x]$ irreducible and $\varphi = \gcd(f \bmod p, g \bmod p)$ in $\mathbb{F}_p[x]$ irreducible of degree n

- 1 Choose some monic polynomial $g_0(x)$ of degree n with small integer coefficients ;
- 2 Choose an integer $W \approx p^{1/(D+1)}$, but slightly larger, and set
 $g(x) = g_0(x + W) = c_0 + c_1 x + \cdots + x^n$;
- 3 Reduce the rows of the following matrix using LLL

$$M = \begin{bmatrix} p & & & \\ & \ddots & & \\ & & p & \\ c_0 & c_1 & \cdots & 1 \\ & \ddots & \ddots & \ddots \\ & c_0 & c_1 & \cdots & 1 \end{bmatrix} \quad \left\{ \begin{array}{l} \deg \varphi = n \\ D + 1 - n \end{array} \right\}, \text{ to get } \text{LLL}(M) = \begin{bmatrix} f_0 & f_1 & \cdots & f_D \\ * & & & \end{bmatrix}.$$

4 **return** $(f = f_D x^D + \cdots + f_0, g, \varphi = g \bmod p)$

Proposition 1. The coefficients of the polynomial f in Algorithm 2 have approximate size $Q^{1/(D+1)}$.

Proof. By construction, $|c_0| \approx W^n \approx Q^{1/(D+1)}$. Since c was chose so that $c_n = 1$, we get $\det(M) = p^n$. The first row of $\text{LLL}(M)$ gives a polynomial f of degree at most D which is divisible by g modulo p . The coefficients of f are of approximate size $(\det M)^{1/(D+1)}$. It is $= p^{n/(D+1)} = Q^{1/(D+1)}$ if we assume that the dimension $D + 1$ stays small.

We can have $\deg(f) = D$ for any value of $D \geq n$. We may want to take $D = n$ for some real-life cases, so that f and g are of degree n ; moreover we take $\varphi \equiv g \pmod{p}$. We give such an example here.

Example 2. Consider again the case of $p = 1000001447$ and $n = 4$ this time. We take g_0 to be a polynomial of degree four and small coefficients, for example $g_0 = x^4 + x^3 + x^2 + x + 1$. We use $W_0 = 64 \geq p^{1/(D+1)}$ and we set

$$g = g_0(x + W_0) = x^4 + 257x^3 + 24769x^2 + 1060993x + 17043521.$$

We construct the lattice of integer polynomials of degree at most 4 which are divisible by g modulo p . Since $D = n$ we are here in a particular case where the lattice corresponds to the line of multiples of g by elements of \mathbb{F}_p . We obtain

$$f = 7791770x^4 + 2481996x^3 - 5928141x^2 + 1465261x + 3462017.$$

Note that f and g have coefficients of size $p^{4/5} = Q^{1/5}$.

3.2 The Generalized Joux–Lercier (GJL) Method¹

Joux and Lercier proposed a method in [15] to select polynomials in the context of prime fields. In Algorithm 3 we generalize their method so that it applies to any finite field.

Algorithm 3. Polynomial selection with the generalized Joux–Lercier method (GJL)

Input: p prime, n integer and $d \geq n$ integer

Output: f, g, φ with $f, g \in \mathbb{Z}[x]$ irreducible and $\varphi = \gcd(f \pmod{p}, g \pmod{p})$ in $\mathbb{F}_p[x]$ irreducible of degree n

- 1 Choose a polynomial $f(x)$ of degree $d + 1$ with small integer coefficients which has a monic irreducible factor $\varphi(x) = \varphi_0 + \varphi_1x + \cdots + x^n$ of degree n modulo p ;
- 2 Reduce the following matrix using LLL

$$M = \left[\begin{array}{cccccc} p & & & & & \\ \ddots & & & & & \\ & p & & & & \\ \varphi_0 & \varphi_1 & \cdots & 1 & & \\ & \ddots & \ddots & \ddots & & \\ & & \varphi_0 & \varphi_1 & \cdots & 1 \end{array} \right] \left\{ \begin{array}{l} \deg \varphi = n \\ d + 1 - n \end{array} \right\} \quad \text{to get } \text{LLL}(M) = \left[\begin{array}{c} g_0 \ g_1 \ \cdots \ g_d \\ * \\ \vdots \end{array} \right].$$

3 return $(f, g = g_0 + g_1x + \cdots + g_dx^d, \varphi)$

¹ Recently, (February 2015), D. Matyukhin informed us that he proposed the same polynomial selection method for his algorithm of discrete logarithm [24], published in Russian.

In the prime field case, where $n = 1$, GJL goes as follows. One starts with a polynomial f of degree $d + 1$ with small coefficients such that f admits a factor φ of degree one, or equivalently a root m modulo p . Then, a matrix M is constructed whose rows correspond to the lattice of polynomials in $\mathbb{Z}[x]$ of degree at most d , that also admit m as a root modulo p . We reduce this matrix with LLL and obtain g from the first row of $\text{LLL}(M)$. Note that one can transform M into M' in Equation (1) by linear combinations on the rows. Hence, $\text{LLL}(M) = \text{LLL}(M')$, so GJL and the original method of Joux–Lercier obtain the same polynomial g .

$$M = \begin{bmatrix} p & 0 & \cdots & 0 \\ -m & 1 & 0 & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & -m & 1 \end{bmatrix}, \quad M' = \begin{bmatrix} p & 0 & \cdots & 0 \\ -m & 1 & 0 & 0 \\ \vdots & \ddots & \ddots & 0 \\ -m^d & \cdots & 0 & 1 \end{bmatrix}, \quad (1)$$

The following result is proved in the same way as Proposition 1.

Proposition 2. *The coefficients of the polynomial g in Algorithm 3 have approximate size $Q^{1/(d+1)}$.*

Remark 1. By considering a smaller matrix, it is possible to produce a polynomial g whose degree is smaller than $d = \deg f - 1$. This does not seem to be a good idea. Indeed, the size of the coefficients of g would be the same as the coefficients of a polynomial obtained starting with a polynomial f with coefficients of the same size but of a smaller degree (d or less).

Example 3. We keep $p = 1000001447$ and $n = 4$ (see Ex. 2). We choose $d = n = 4$, $f = x^5 + x^4 - 7x^3 + 4x^2 + x + 1$ of degree 5. Then f factors modulo p and has a degree four factor $\varphi = x^4 + 234892989x^3 + 208762833x^2 + 670387270x + 109760434$. We construct the lattice of polynomials of degree at most 4 which are divisible by φ modulo p . Reducing it, we obtain

$$g = 8117108x^4 + 1234709x^3 + 9836008x^2 - 1577146x + 7720480 .$$

The polynomial f has coefficients of size $O(1)$; g of size $O(p^{4/5})$. More precisely, $\log_2 p = 30$ and $\log_2 \|g\|_\infty = 24 = 4/5 \log_2 p$.

3.3 The Conjugation Method

Roughly speaking, the aim of polynomial selection is to produce two polynomials f, g such that the norm of f , resp. g evaluated at some algebraic integer $a - b\alpha$ (or $\phi(\alpha)$) is minimal, with respect to the size of a and b (see Sec. 2). This means, the degrees of f and g need to stay small but also the size of their coefficients. The previous method GJL was focused on minimizing the degrees of both f and g to e.g. n and $n + 1$. Here we set f with a higher degree: $2n$ instead of $n + 1$, so that the size of the coefficients of g is always $O(\sqrt{p})$. We cannot achieve a so small

coefficient size for g with the GJL method. We show in Sec. 4.4 that despite the higher degree of f , this method is better in practice than any other one when $n = 2, 3$ and p is more than 660 bits, and in particular for cryptographic sizes.

Let us give the idea of our method. We first carefully select f of degree $2n$, irreducible over \mathbb{Z} , and so that it factors into two irreducible conjugate polynomials φ and $\bar{\varphi}$ over some quadratic extension of \mathbb{Q} . If we can embed this quadratic extension in \mathbb{F}_p , we end up with two irreducible factors of f modulo p . Because of our judicious choice of f with two conjugate factors, we obtain g whose coefficients have size $O(\sqrt{p})$ using rational reconstruction. We give a first example in Ex. 4 to clarify what we have in mind.

We start with two examples, and then give the general construction.

Example 4. We target \mathbb{F}_{p^4} with $p = 1000010633$ and $n = 4$. We try integers $a = -2, -3, \dots$ until a is a square but not a fourth power in \mathbb{F}_p . We find $a = -9$. We set $f = x^8 - a = x^8 + 9$ which is irreducible over \mathbb{Z} . Observe that by construction, f has two degree 4 conjugate irreducible factors $f = (x^4 - \sqrt{a})(x^4 + \sqrt{a})$. We set $\varphi = x^4 - \sqrt{a}$ which, due to the choice of a , belongs to $\mathbb{F}_p[x]$ and is irreducible. We continue by computing a rational reconstruction (u, v) of \sqrt{a} modulo p : $u \cdot v^{-1} \equiv \sqrt{a} \pmod{p}$; here $u = -58281$ and $v = 24952$. Finally we set $g = vx^4 - u = 24952x^4 + 58281$ of norm $\|g\|_\infty \sim \sqrt{p}$. Note that we respect the condition $\varphi = \gcd(f \pmod{p}, g \pmod{p})$.

In the previous example, the prime p was given and we searched for a parameter a , or equivalently for a polynomial $f = x^8 - a$. It turns out that some polynomials f have very good sieving properties and we would like to use them for many primes p . In this case, we can reverse the process and start with a given f , while expecting to succeed in only a fraction of the cases.

Example 5. We want to use $f = x^4 + 1$. We target \mathbb{F}_{p^2} for $p \equiv 7 \pmod{8}$ prime. Note that $f(x) = (x^2 + \sqrt{2}x + 1)(x^2 - \sqrt{2}x + 1)$ over $\mathbb{Q}(\sqrt{2})$. Since 2 is a square modulo p , we take $\varphi = x^2 + \sqrt{2}x + 1 \in \mathbb{F}_p[x]$ and note that φ is a factor of f modulo p . Now, by rational reconstruction of $\sqrt{2}$ in \mathbb{F}_p , we can obtain two integers $u, v \in \mathbb{Z}$ such that $\frac{u}{v} \equiv \sqrt{2} \pmod{p}$, and u and v have size similar to \sqrt{p} . We define $g = vx^2 + ux + v$. Then f and g share a common irreducible factor of degree 2 modulo p , and satisfy the degree and size properties given in Prop. 3.

Although the technique in the second example is more interesting in practice, it is the construction in the first example that can be made general, as given in Algorithm 4. Under reasonable assumptions, this algorithm terminates and finds pairs of polynomials f and g with the claimed degree and size properties for any extension field \mathbb{F}_{p^n} .

Proposition 3. *The polynomials (f, g) returned by Algorithm 4 satisfy the following properties*

1. *f and g have integer coefficients and degrees $2n$ and n respectively;*
2. *the coefficients of f have size $O(1)$ and the coefficients of g are bounded by $O(\sqrt{p})$;*

Algorithm 4. Polynomial selection with the Conjugation method (Conj)

Input: p prime and n integer
Output: f, g, φ with $f, g \in \mathbb{Z}[x]$ irreducible and $\varphi = \gcd(f \bmod p, g \bmod p)$ in $\mathbb{F}_p[x]$ irreducible of degree n

- 1 **repeat**
- 2 Select $g_1(x), g_0(x)$, two polynomials with small integer coefficients,
 $\deg g_1 < \deg g_0 = n$;
- 3 Select $\mu(x)$ a quadratic, monic, irreducible polynomial over \mathbb{Z} with small
coefficients ;
- 4 **until** $\mu(x)$ has a root λ in \mathbb{F}_p and $\varphi = g_0 + \lambda g_1$ is irreducible in $\mathbb{F}_p[x]$;
- 5 $(u, v) \leftarrow$ a rational reconstruction of λ ;
- 6 $f \leftarrow \text{Res}_Y(\mu(Y), g_0(x) + Yg_1(x))$;
- 7 $g \leftarrow vg_0 + ug_1$;
- 8 **return** (f, g, φ)

3. f and g have a common irreducible factor φ of degree n over \mathbb{F}_p .

Proof. The polynomial f is the resultant of two bivariate polynomials with integer coefficients. Using classical properties of the resultant, f can be seen as the product of the polynomial $g_0(x) + Yg_1(x)$ evaluated in Y at the two roots of $\mu(Y)$, therefore its degree is $2n$. Since all the coefficients of the polynomials involved in the definition of f have size $O(1)$, and the degree n is assumed to be “small”, then the coefficients of f are also $O(1)$. For the size of the coefficients of g , it follows from the output of the rational reconstruction of λ in \mathbb{F}_p , which is expected to have sizes in $O(\sqrt{p})$. The polynomials f and g are suitable for NFS in \mathbb{F}_{p^n} , because both are divisible by $\varphi = g_0 + \lambda g_1$ modulo p , and by construction φ is irreducible of degree n .

In the example above (Ex. 5), for \mathbb{F}_{p^2} with $p \equiv 7 \pmod{8}$, Algorithm 4 was applied with $g_1 = x$, $g_0 = x^2 + 1$ and $\mu = x^2 - 2$. One can check that $f = \text{Res}_Y(Y^2 - 2, (x^2 + 1) + Yx) = x^4 + 1$.

In the following section, an asymptotic analysis shows that there are cases where this Conjugation method is more interesting than JLSV₁ and that GJL is competitive with JLSV₂; furthermore, the new methods are also well-suited for small degree extensions that can be reached with current implementations.

4 Complexity Analysis

In this section we prove Theorem 1 that we stated in the Introduction.

4.1 Preliminaries

As introduced in Section 2, the parameter t denotes the number of terms of the polynomials in the sieving domain, i.e. $\deg \phi = t - 1$, and B is the smoothness bound. We call E the square root of the number of sieved polynomials, i.e. the

coefficients of ϕ belong to the interval $[-E^{2/t}, E^{2/t}]$. Kalkbrener's bound [19, Corollary 2] states that, for any polynomials f and ϕ ,

$$|\text{Res}(f, \phi)| \leq \kappa(\deg f, \deg \phi) \|f\|_\infty^{\deg \phi} \|\phi\|_\infty^{\deg f},$$

where $\kappa(n, m) = \binom{n+m}{n} \binom{n+m-1}{n}$. For two polynomials f and g we write $\kappa(f, g)$ for $\kappa(\deg f, \deg g)$. Hence, we obtain a bound on the product of the norms:

$$|\text{Res}(f, \phi) \text{Res}(g, \phi)| \leq \kappa(f, \phi) \kappa(g, \phi) \|\phi\|_\infty^{\deg f + \deg g} (\|f\|_\infty \|g\|_\infty)^{\deg \phi}. \quad (2)$$

A simple upper bound for $\kappa(n, m)$ is $(n+m)!$:

$$\begin{aligned} |\text{Res}(f, \phi) \text{Res}(g, \phi)| &\leq \\ &\leq (\deg f + \deg \phi)! (\deg g + \deg \phi)! (\|f\|_\infty \|g\|_\infty)^t \|\phi\|_\infty^{\deg f + \deg g} \\ &\leq (\deg f + t - 1)! (\deg g + t - 1)! (\|f\|_\infty \|g\|_\infty)^{t-1} E^{2(\deg f + \deg g)/t}. \end{aligned}$$

In what follows we respect make sure that the degrees of our polynomials satisfy: $\deg f + \deg g + t = O(1) \max((\log Q)^{1/3}, n)$. Writing $p = L_Q(\alpha)$ with $\alpha > 1/3$, we obtain $n = O(1)(\log Q)^{1-\alpha}/(\log \log Q)^{1-\alpha}$. Then, we have $(\deg f + t - 1)! (\deg g + t - 1)! = L_Q(\max(1 - \alpha, 1/3))$, whereas our estimates for the right hand side will have a size $L_Q(2/3)$. This allows us to use the estimation

$$\max_{\phi \text{ in sieving domain}} |\text{Res}(f, \phi) \text{Res}(g, \phi)| \approx (\|f\|_\infty \|g\|_\infty)^{t-1} E^{2(\deg f + \deg g)/t}. \quad (3)$$

Since the number of sieved polynomials is E^2 , the cost of the sieve is $E^{2+o(1)}$. Independently of the choice of the polynomials f and g , the cardinality of the factor base is $B^{1+o(1)}$, and using the (block) Wiedemann algorithm [10, 33], the cost of the linear algebra is $B^{2+o(1)}$. Hence, we set $E = B$ and, since we expect an algorithm of complexity $L_Q(1/3)$, the two are equal to $L_Q(1/3, \beta)$

$$E = B = L_Q(1/3, \beta)$$

for a constant β to be found.

We make the common heuristic that the product of the resultants of the polynomials ϕ in the sieving domain with f and g has the same probability to be B -smooth as a random integer of the same size; we denote this probability by \mathcal{P} . Since the cost of the sieve is $B\mathcal{P}^{-1}$ and, at the same time $E^{2+o(1)}$, we find the equation

$$B = \mathcal{P}^{-1}. \quad (4)$$

4.2 The Generalized Joux–Lercier Method

We are now ready to prove the first part of Theorem 1. Using GJL, one constructs two polynomials f and g such that, for a parameter $d \geq n$, we have $\deg f = d+1$, $\deg g = d$, $\|g\|_\infty \approx Q^{1/(d+1)}$ and $\|f\|_\infty$ of size $O(1)$.

The GJL polynomials have the same degree and coefficient size as those obtained in [15] for prime fields. Hence, we make the same choices for parameter

t , i.e. we sieve on linear polynomials. Also, we take d of roughly the same size, i.e. $d = \delta((\log Q)/(\log \log Q))^{1/3}$, which we inject in Equation (3):

$$\max_{\phi} (|\operatorname{Res}(f, \phi) \operatorname{Res}(g, \phi)|) \approx \|f\|_{\infty} \|g\|_{\infty} E^{\deg f + \deg g}, \\ \approx Q^{1/(d+1)} E^{2d+1}.$$

With the L -notation, we obtain

$$|\operatorname{Res}(f, \phi) \operatorname{Res}(g, \phi)| \leq L_Q \left(2/3, \delta\beta + \frac{2}{\delta} \right).$$

Using the Canfield–Erdős–Pomerance theorem [7], we obtain

$$\mathcal{P} = 1/L_Q \left(1/3, \frac{\delta}{3} + \frac{2}{3\beta\delta} \right).$$

The equality $\mathcal{P}^{-1} = B$ imposes

$$\beta = \frac{\delta}{3} + \frac{2}{3\beta\delta}.$$

The optimal value of δ is the one which minimizes the expression on the right hand side, so we take $\delta = \sqrt{2/\beta}$ and we obtain $\beta = 2/3\sqrt{2/\beta}$, or equivalently $\beta = \sqrt[3]{8/9}$. Since the complexity of NFS is $(E^2 + B^2)^{1+o(1)} = L_Q(1/3, 2\beta)$, we obtain the complexity given in Theorem 1.

The range of application of GJL is determined by the condition $n \leq d$. This is true for all fields of large characteristic. In the boundary case, since $d = \delta/2 \left(\frac{\log Q}{\log \log Q} \right)^{1/3}$ with $\delta = \sqrt{2/\beta} = \sqrt[3]{3}$, the GJL method applies only to those fields \mathbb{F}_{p^n} such that

$$p \geq L_Q \left(2/3, \sqrt[3]{8/3} \right).$$

This concludes the proof of the first part of Theorem 1.

4.3 The Conjugation Method

Recall that the Conjugation method allows us to construct two polynomials f and g such that $\deg f = 2n$, $\deg g = n$, $\|g\|_{\infty} \approx Q^{1/(2n)}$ and $\|f\|_{\infty} = O(1)$. When introduced in Equation (3), these values give

$$|\operatorname{Res}(\phi, f) \operatorname{Res}(\phi, g)| \leq \left(E^{6n/t} Q^{(t-1)/2n} \right)^{1+o(1)}. \quad (5)$$

We study first the case of medium characteristic and then the boundary case between medium and large characteristic.

The Case of Medium Characteristic. The Conj polynomials are similar to the ones in [16] obtained using JLSV_1 , so we choose parameters of the same type as in [16] and set

$$t = c_t n \left(\frac{\log Q}{\log \log Q} \right)^{-1/3}.$$

Using $E = B = L_Q(1/3, \beta)$ in Equation (5), the product of the two resultants has norm $L_Q(2/3, 6\beta/c_t + c_t/2)$. Due to the Canfield–Erdős–Pomerance theorem, the probability that a polynomial ϕ in the sieving domain has B -smooth resultants is

$$\mathcal{P} = 1/L_Q \left(1/3, \frac{2}{c_t} + \frac{c_t}{6\beta} \right).$$

We choose $c_t = 2\sqrt{3\beta}$ in order to optimize this probability:

$$\mathcal{P} = 1/L_Q \left(1/3, 2/\sqrt{3\beta} \right).$$

From the condition $\mathcal{P}^{-1} = B$, we have $\beta = \sqrt[3]{4/3}$, and we obtain the second result in Theorem 1.

The Boundary Case. For every constant $c_p > 0$, we consider the family of finite fields \mathbb{F}_{p^n} such that

$$p = L_{p^n}(2/3, c_p)^{1+o(1)}.$$

We will take parameter t (Section 2) to be a constant in this analysis. Then the probability that a polynomial ϕ in the sieving domain has B -smooth resultants is

$$\mathcal{P} = 1/L_Q \left(1/3, \frac{2}{c_p t} + \frac{c_p(t-1)}{6\beta} \right).$$

The condition $\mathcal{P}^{-1} = B$ leads to $\frac{2}{c_p t} + \frac{c_p(t-1)}{6\beta} = \beta$, or equivalently to $\beta = \frac{1}{c_p t} + \sqrt{\frac{1}{(c_p t)^2} + \frac{1}{6} c_p(t-1)}$.

This completes the proof of the following result.

Proposition 4. *When $\log Q$ goes to infinity and p satisfies the condition $p = L_Q(2/3, c_p)$, the complexity of NFS with the Conjugation method is:*

$$L_Q \left(1/3, \frac{2}{c_p t} + \sqrt{\frac{4}{(c_p t)^2} + \frac{2}{3} c_p(t-1)} \right).$$

In Figure 1 we have plotted the complexities of Proposition 4, together with GJL and the Multiple number field sieve variant of [3].² There are some ranges of the parameter c_p where the Conjugation method is the fastest and a range where the GJL method is optimal. The best case for NFS with Conjugation polynomials corresponds to $c_p = 12^{1/3} \approx 2.29$ and $t = 2$. In this case we get the third result in Theorem 1.

² Thanks to Cécile Pierrot who pointed to us that this figure was inexact in an earlier variant of the manuscript.

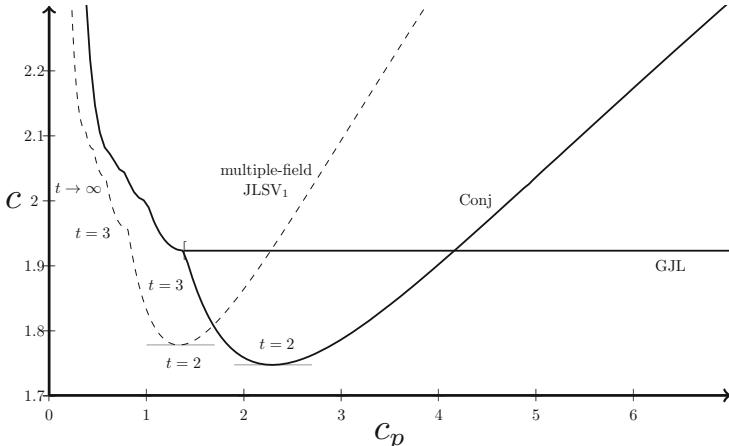


Fig. 1. The complexity of NFS for fields \mathbb{F}_{p^n} with $p = L_{p^n}(2/3, c_p)$ is $L_{p^n}(1/3, c)$

4.4 Non-asymptotic Comparisons for Small Extension Degrees

Let us make a practical (as opposed to asymptotic) analysis of the four methods in our arsenal:

- the methods of Joux, Lercier, Smart and Vercauteren: JLSV₁ and JLSV₂.
- the generalized Joux–Lercier method, GJL; to indicate the value of the parameter d we sometimes write GJL-($d+1, d$).
- the Conjugation method, Conj.

We do not include Schirokauer's variant in our study since it is very different in nature, requiring to sieve on polynomials with coefficients in small degree extensions of \mathbb{Q} .

The complexity analysis that was presented earlier in this section gives hints, but does not allow us to choose the best method. For example, if one wants to compute discrete logarithms in finite fields \mathbb{F}_Q of constant bitsize, i.e. $\log_2 Q \approx \text{const}$, then JLSV₁ and Conj are competitive when p is smaller (medium prime case), whereas JLSV₂ and GJL are better when p is larger (large characteristic case). Also, we expect the choice $t = 2$ to be optimal when p is large, whereas we might consider sieving on non-linear polynomials, i.e. $t \geq 3$, for smaller values of p .

Table 1 summarizes the properties of the polynomials obtained with each method.

Although JLSV₂ was the state-of-art for the non-prime fields of large characteristic, it is now beaten either by GJL or by JLSV₁:

Table 1. Summary of the sizes of the norms product corresponding to various methods. Here \mathbb{F}_Q , $Q = p^n$, is the target field, d and D are parameters in the polynomial selection stage and E is the sieve parameter.

method	$\deg f$	$\deg g$	$\ f\ _\infty$	$\ g\ _\infty$	product of norms
Conj	$2n$	n	$O(1)$	$Q^{1/(2n)}$	$E^{6n/t} Q^{(t-1)/(2n)}$
GJL	$d+1$	$d \geq n$	$O(1)$	$Q^{1/(d+1)}$	$E^{2(2d+1)/t} Q^{(t-1)/(d+1)}$
JLSV ₁	n	n	$Q^{1/(2n)}$	$Q^{1/(2n)}$	$E^{4n/t} Q^{(t-1)/n}$
JLSV ₂	$D \geq n$	n	$Q^{1/(D+1)}$	$Q^{1/(D+1)}$	$E^{2(D+n)/t} Q^{2(t-1)/(D+1)}$

Proposition 5. Let n, t, D be integers with $n, t \geq 2$ and $D \geq n$, and let E and Q be positive real numbers. Then the quantity $(E^{n+D})^{2/t} (Q^{2/(D+1)})^{t-1}$ is either larger than $(E^{2n})^{2/t} (Q^{1/n})^{t-1}$ or we can select $d \geq n$ so that it is larger than $(E^{2d+1})^{2/t} (Q^{1/(d+1)})^{t-1}$.

Proof. **Case** $D \geq 2n$. We set $d = \lfloor D/2 \rfloor$ and we use GJL with this parameter. On the one hand we have $2(\lfloor D/2 \rfloor + 1) \geq D + 1$, so $(Q^{2/(D+1)})^{t-1} \geq (Q^{1/(d+1)})^{t-1}$. On the other hand, we have $n+D \geq 1+(D+1) \geq 1+(2\lfloor D/2 \rfloor + 1)$, so $(E^{n+D})^{2/t} \geq (E^{2d+1})^{2/t}$.

Case $n \leq D \leq 2n-1$. On the one hand we have $D+n \geq 2n$, so $(E^{n+D})^{2/t} \geq (E^{2n})^{2/t}$. On the other hand, $2/(D+1) \geq 1/n$, so $(Q^{2/(D+1)})^{t-1} \geq (Q^{1/n})^{t-1}$.

In order to compare the remaining candidates we need to plug numerical values into Equation (2). The parameter E is hard to determine, and depends on the polynomials which are used. For example, better polynomials allow us to sieve less and hence to use a smaller value of E . Luckily, the difference between the various values of E are not very large and, when one method is considerably better than another, an approximate value of E is enough for the comparison. Another bias we are aware of is that for norm products of similar sizes, a method that provides norms that are well-balanced should be better than if the norm on one side is much larger than the norm on the other side. Therefore, when the differences between methods are small, we cannot decide by looking only at the size of the norm product. Table 2 lists the values of E with respect to the cardinality Q of the target field, obtained from the default parameters of the CADO factoring software [1] up to $\log_{10} Q = 220$ and extrapolated afterwards. But these values should not be taken too seriously in order to derive security parameters. The goal is only to investigate the relative performances of the various methods of polynomial selection for sizes where it is currently too costly to do experiments.

We considered several values of parameters d and $t = \deg \phi + 1$. As the asymptotic analysis predicts for the case of large characteristic (small degree), our results showed that the choice $t = 2$ is optimal for $n = 2, 3, 4, 5$. Since, in addition, the comparison goes in a similar manner for each value of parameter t , we focus on the case $t = 2$. We make an exception to this rule for $n = 6$, where the choice $t = 3$ is optimal for some ranges of $\log_{10} Q$.

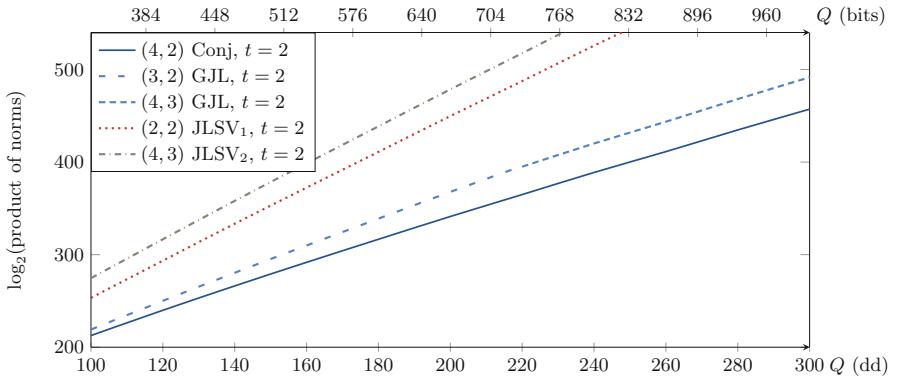
Table 2. Practical values of E for Q from 100 to 300 decimal digits(dd)

$Q(\text{dd})$	100	120	140	160	180	200	220	240	260	280	300
$Q(\text{bits})$	333	399	466	532	598	665	731	798	864	931	997
$E(\text{bits})$	20.9	22.7	24.3	25.8	27.2	28.5	29.7	30.9	31.9	33.0	34.0

In Table 3 we expand the formula of Equation (3) for $t = 2$ and $n = 2, 3, 4, 5, 6$. Note also that we list several values for the parameter d of GJL since it cannot be fixed by asymptotic arguments. Using the remark that the quotient $\log Q / \log E$ belongs to the interval $[15, 30]$, we discard some choices, and mark them with an \otimes . For example, JLSV₁ beats GJL-(3, 2) only when $\log Q / \log E \leq 6$, which is outside of our range of interest.

4.5 Final Results

The Case $n = 2$. We draw the curves corresponding to the results in Table 3, since $t = 2$ is optimal in this case. From Figure 2, the best choice is to use Conj polynomials.

**Fig. 2.** Polynomials for \mathbb{F}_{p^2}

The Case $n = 3$. Again, it is optimal to sieve on linear polynomials. For smaller values, i.e. Q of less than 220 decimal digits, we use GJL-(4, 3) polynomials, whereas for larger fields we switch to Conj, as exemplified in Figure 3.

The Case $n = 4$. We sieve on linear polynomials as before ($t = 2$). Here JLSV₁ and GJL-(5, 4) offer similar polynomials, see Figure 4.

The Case $n = 5$. Several methods give polynomials with a norm product of roughly the same size: Conj with $t = 3$ and $t = 4$, GJL-(6, 5) with $t = 2$ and $t = 3$ and, finally JLSV₁ with $t = 2$. All this is demonstrated in Figure 5.

Table 3. Size of the product of norms for various methods and associated parameters, when one sieves on linear polynomials ($t = 2$). We discard (\otimes) the methods which offer sizes of norms product which are clearly not competitive compared to some other one, assuming that $15 \leq \log Q / \log E \leq 30$ (Tab. 2).

n	method	$(\deg g, \deg f)$	$\ f\ _\infty$	$\ g\ _\infty$	$E^{\deg f + \deg g} \ f\ _\infty \ g\ _\infty$	discard
2	GJL - (3, 2)	(3, 2)	$O(1)$	$Q^{1/3}$	$E^5 Q^{1/3}$	
	GJL - (4, 3)	(4, 3)		$Q^{1/4}$	$E^7 Q^{1/4}$	\otimes
	Conj	(4, 2)		$Q^{1/4}$	$E^6 Q^{1/4}$	
	JLSV ₁	(2, 2)		$Q^{1/4}$	$E^4 Q^{1/2}$	\otimes
n	method	$(\deg f, \deg g)$	$\ f\ _\infty$	$\ g\ _\infty$	$E^{\deg f + \deg g} \ f\ _\infty \ g\ _\infty$	discard
3	GJL - (4, 3)	(4, 3)	$O(1)$	$Q^{1/4}$	$E^7 Q^{1/4}$	
	GJL - (5, 4)	(5, 4)		$Q^{1/5}$	$E^9 Q^{1/5}$	\otimes
	Conj	(6, 3)		$Q^{1/6}$	$E^9 Q^{1/6}$	
	JLSV ₁	(3, 3)		$Q^{1/6}$	$E^6 Q^{1/3}$	\otimes
n	method	$(\deg f, \deg g)$	$\ f\ _\infty$	$\ g\ _\infty$	$E^{\deg f + \deg g} \ f\ _\infty \ g\ _\infty$	discard
4	GJL - (5, 4)	(5, 4)	$O(1)$	$Q^{1/5}$	$E^9 Q^{1/5}$	
	GJL - (6, 5)	(6, 5)		$Q^{1/6}$	$E^{11} Q^{1/6}$	\otimes
	Conj	(8, 4)		$Q^{1/8}$	$E^{12} Q^{1/8}$	\otimes
	JLSV ₁	(4, 4)		$Q^{1/8}$	$E^8 Q^{1/4}$	
n	method	$(\deg f, \deg g)$	$\ f\ _\infty$	$\ g\ _\infty$	$E^{\deg f + \deg g} \ f\ _\infty \ g\ _\infty$	discard
5	GJL - (6, 5)	(6, 5)	$O(1)$	$Q^{1/6}$	$E^{11} Q^{1/6}$	
	GJL - (7, 6)	(7, 6)		$Q^{1/7}$	$E^{13} Q^{1/7}$	\otimes
	Conj	(10, 5)		$Q^{1/10}$	$E^{15} Q^{1/10}$	\otimes
	JLSV ₁	(5, 5)		$Q^{1/10}$	$E^{10} Q^{1/5}$	
n	method	$(\deg f, \deg g)$	$\ f\ _\infty$	$\ g\ _\infty$	$E^{\deg f + \deg g} \ f\ _\infty \ g\ _\infty$	discard
6	GJL - (7, 6)	(7, 6)	$O(1)$	$Q^{1/7}$	$E^{13} Q^{1/7}$	
	GJL - (8, 7)	(8, 7)		$Q^{1/8}$	$E^{15} Q^{1/8}$	\otimes
	Conj	(12, 6)		$Q^{1/12}$	$E^{18} Q^{1/12}$	\otimes
	JLSV ₁	(6, 6)		$Q^{1/12}$	$E^{12} Q^{1/6}$	

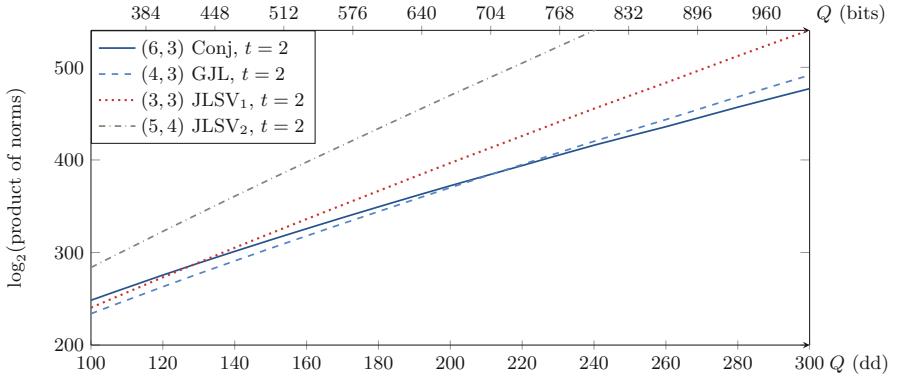


Fig. 3. Polynomials for \mathbb{F}_{p^3}

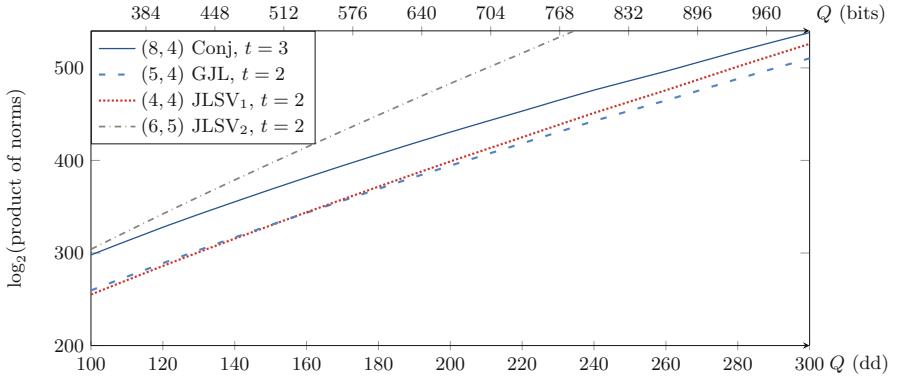


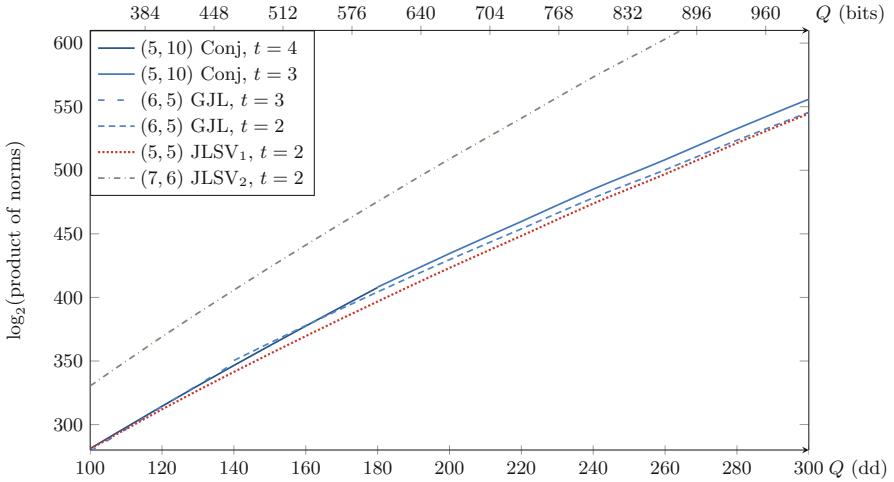
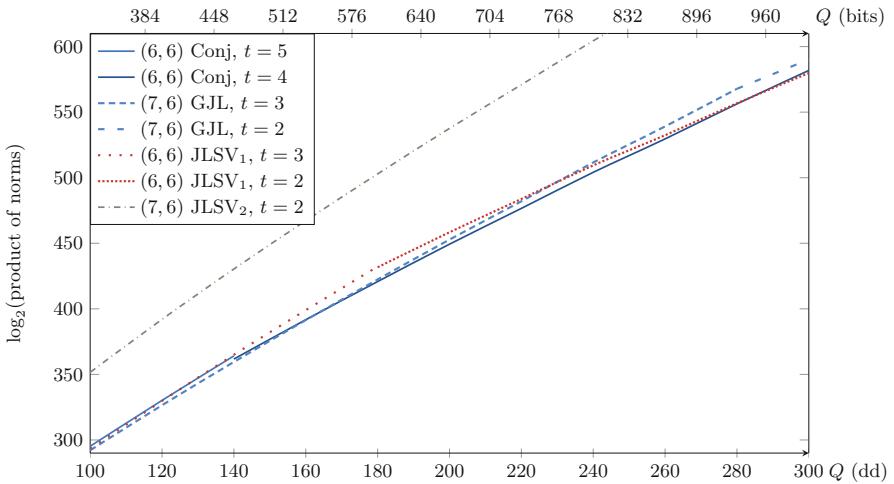
Fig. 4. Polynomials for \mathbb{F}_{p^4}

The Case $n = 6$. When Q is less than 180 decimal digits, the choice is between GJL-(7,6) with $t = 3$ and Conj with $t = 4$. When Q is larger than 260 decimal digits, the best two methods are Conj with $t = 4$ and JLSV₁ with $t = 2$. Between the two ranges, one needs to consider the three methods listed before. See Figure 6.

5 Additional Improvements

5.1 Improving the Root Properties

In both GJL and Conjugation methods, it is possible to obtain more than one polynomial g for a given f by taking another choice for the rational reconstruction, or another vector in the reduced lattice. Hence, we can assume that one has obtained two distinct reduced polynomials g_1 and $g_2 \in \mathbb{Z}[x]$ such that φ

**Fig. 5.** Polynomials for \mathbb{F}_{p^5} **Fig. 6.** Polynomials for \mathbb{F}_{p^6}

divides both g_1 and g_2 in $\mathbb{F}_p[x]$. Any linear combination $g = \lambda_1 g_1 + \lambda_2 g_2$ for small integers λ_1 and λ_2 is then suitable for running the NFS algorithm.

The Murphy \mathbb{E} value as explained in [25, Sec. 5.2.1, Eq. 5.7 p. 86] is a good criterion to choose between all these g polynomials. In our experiments we searched for $g = \lambda_1 g_1 + \lambda_2 g_2$ with $|\lambda_i| < 200$ and such that $\mathbb{E}(f, g)$ is maximal. In practice we obtain g with $\alpha(g) \leq -1.5$ and $\mathbb{E}(f, g)$ improved by 2% up to 30 %.

5.2 Coppersmith's Multiple-field Variant

In [9], Coppersmith introduced a variant of NFS in which more than two polynomials are used. This can be applied to essentially all polynomial selection methods and in particular to the ones mentioned in this article. The base- m method, which applies only to prime fields, was analyzed by Matyukhin [23]. Recently, it was shown that JLSV₁ and JLSV₂ can successfully be adapted to use multiple fields, as demonstrated by [3].

Another important example is the Conjugation method. In Sec. 5.1, we noted that the same polynomial f can be paired with any of the two polynomials g_1 and g_2 , and that we have $\gcd(f \bmod p, g_1 \bmod p, g_2 \bmod p) = \varphi$. This fact can be used to derive a multiple-field variant. It was remarked and analyzed in [26] and results in a complexity of $L_Q(1/3, c)$ with $c = (8(9 + 4\sqrt{6})/15)^{1/3} \approx 2.156$, in the medium characteristic case.

We have also analyzed a multiple-field variant of the Generalized Joux–Lercier method. This provides only a marginal improvement in the theoretical complexity and is not competitive with other methods [3], and therefore we do not include this analysis here.

5.3 Taking Advantage of Automorphisms

Joux, Lercier, Smart and Vercauteren [16, Section 4.3] proposed to speed up computations in NFS using number field automorphisms. Given a field K and an irreducible polynomial $f \in K[x]$ without multiple roots, a K -automorphism is a rational fraction $A \in K(x)$ such that, for some rational fraction $D(x) \in K(x)$, we have $f(A(x)) = D(x)f(x)$. Using the language of Galois theory, a K -automorphism is an automorphism of the extension $(K[x]/\langle f \rangle)/K$. Hence, the automorphisms form a group whose order divides $\deg f$.

It is possible to push further the idea in [16] so that one can use automorphisms of both polynomials f and g . An example is given in our record computation described in Section 6 where we used two reciprocal polynomials: this saves a factor of two in the sieve and a factor of four in the linear algebra.

When a method to select polynomials gives the choice of the first polynomial f , we can select f in the family of our preference, making possible for example to have automorphisms. The only obstacle is that we cannot find polynomials with an automorphism of order n , as required by the results in [16] if $\deg f$ is not a multiple of n .

But in fact some methods allow us to have automorphisms for both polynomials. Indeed, the literature, e.g. [11], offers examples of polynomials $g_0, g_1 \in \mathbb{Q}[x]$ and rational fractions $A(x) \in \mathbb{Q}(x)$ such that, for any number field K and any parameter $a \in K$, the polynomial $g_0 + ag_1$ admits A as a K -automorphism:

$$\exists D_a(x) \in K(x), \quad g_0(A(x)) + ag_1(A(x)) = D_a(x)(g_0(x) + ag_1(x)).$$

Example 6. For $g_0 = x^3 - 3x - 1$ and $g_1 = x^2 + x$, the rational fraction $A = -(1+1/x)$ is an automorphism, for any value of parameter a . Indeed, $g_0(A(x)) + ag_1(A(x))$ is $(x^3 + ax^2 + (a-3)x - 1) = (g_0(x) + ag_1(x))/x^3$, so $D(x) = 1/x^3$.

We do not study the question of finding such families. Let us instead make a list of the cases where one or both polynomials can admit automorphisms.

- JLSV₁ allows both polynomials to be in a family of type $g_0 + ag_1$, with g_0 and g_1 fixed; we can have automorphisms on both sides.
- JLSV₂ allows g to be selected with good properties; since $\deg g = n$, g can have \mathbb{Q} -automorphisms of order n .
- GJL allows f to be selected in the family of our choice; when $\deg f$ is divisible by n , f can have \mathbb{Q} -automorphisms of order n .
- Conj allows us to have \mathbb{Q} -automorphisms for both polynomials, for the values of n where families as above can be found. On the one hand, g is chosen in the family $\{g_0 + ag_1\}$, of automorphism $A(x)$. On the other hand, let ω be an algebraic number, root of an irreducible degree two polynomial $\mu \in \mathbb{Q}[x]$, such that $f = (g_0 + \omega g_1)(g_0 + \bar{\omega} g_1)$. Let $D_\omega \in \mathbb{Q}(\omega)(x)$ be such that $g_0(A(x)) + \omega g_1(A(x)) = D_\omega(x)(g_0(x) + \omega g_1(x))$. Then, by conjugation in $\mathbb{Q}(\omega)$ we have $g_0(A(x)) + \bar{\omega} g_1(A(x)) = \overline{D_\omega}(x)(g_0(x) + \bar{\omega} g_1(x))$. When we multiply, we get

$$\begin{aligned} f(A(x)) &= (g_0(A(x)) + \omega g_1(A(x))) \cdot (g_0(A(x)) + \bar{\omega} g_1(A(x))) \\ &= (D_\omega(x)(g_0(x) + \omega g_1(x))) \cdot (\overline{D_\omega}(x)(g_0(x) + \bar{\omega} g_1(x))) \\ &= (D_\omega(x)\overline{D_\omega}(x)) f(x). \end{aligned}$$

By noting that $D_\omega(x)\overline{D_\omega}(x)$ belongs to $\mathbb{Q}(x)$, we conclude that A is a \mathbb{Q} -automorphism for f .

6 Record Computations

6.1 Setup

In order to test how our ideas perform in practice, we did several medium-sized practical experiments in fields of the form \mathbb{F}_{p^2} . We have decided to choose a prime number p of 90 decimal digits so that \mathbb{F}_{p^2} has size 180 digits, the current record-size for \mathbb{F}_p . This corresponds to a 600-bit field. To demonstrate that our approach is not specific to a particular form of the prime, we took the first 90 decimal digits of π . Our prime number p is the next prime such that $p \equiv 7 \pmod{8}$ and both $p+1$ and $p-1$ have a large prime factor: $p = \lfloor \pi \cdot 10^{89} \rfloor + 14905741$.

$$\begin{aligned} p &= 3141592653589793238462643383279502884197169399375105820974 \\ &\quad 94459230781640628620899877709223 \\ \ell &= 3926990816987241548078304229099378605246461749218882276218 \\ &\quad 6807403847705078577612484713653 \end{aligned}$$

$$p-1 = 6 \cdot h_0 \text{ with } h_0 \text{ a 89 digit prime}$$

$$p+1 = 8 \cdot \ell$$

We solved the discrete logarithm problem in the order ℓ subgroup. We imposed p to be congruent to -1 modulo 8, so that the polynomial $f(x) = x^4 + 1$ could be used. The Conjugation method yields a polynomial g of degree 2 and negative discriminant:

$$\begin{aligned} f &= x^4 + 1 \\ g &= 448225077249286433565160965828828303618362474 \ x^2 \\ &\quad - 296061099084763680469275137306557962657824623 \ x \\ &\quad + 448225077249286433565160965828828303618362474 . \end{aligned}$$

Since p is 90 digits long, the coefficients of g have 45 digits. The polynomials f and g have the irreducible factor

$$\varphi = t^2 + 10778151309582301866698988310224439480941229764389534 \backslash \\ 9097410632508049455376698784691699593 t + 1$$

in common modulo p , and \mathbb{F}_{p^2} will be taken as $\mathbb{F}_p[X]/\langle \varphi \rangle$.

This choice of polynomials has several practical advantages. Both f and g are reciprocal polynomials, so that $x \mapsto 1/x$ is an automorphism in the sense of Subsection 5.3. This provides a speed-up by a factor of 2 for the relation collection and a factor of 4 in the linear algebra, as explained below. Furthermore, the polynomial f corresponds to a number field K_f with unit rank 1, and a fundamental unit is given by the fundamental unit of the subfield $\mathbb{Q}(\sqrt{2})$. By construction, 2 is a square modulo p , so that $\sqrt{2}$ belongs to \mathbb{F}_p . Then, the image in \mathbb{F}_{p^2} of the fundamental unit of K_f is actually in \mathbb{F}_p and its discrete logarithm is 0 modulo ℓ . Since the polynomial g corresponds to a number field with unit rank 0, we do not need Schirokauer maps for this case. Generalizations of this interesting fact will be explained elsewhere.

6.2 Collecting Relations

The relation collection step was then done using the sieving software of CADO-NFS [1]. More precisely, we used the special- \mathfrak{q} technique for ideals \mathfrak{q} on the g -side, since it produces norms that are larger than on the f -side. We sieved all the special- \mathfrak{q} s between 120,000,000 and 160,000,000, keeping only one in each pair of conjugates under the action $x \mapsto 1/x$. Indeed, if $\phi = a - bx$ gives a relation for a special- \mathfrak{q} , then $b - ax$ yields a relation for the conjugate ideal of this special- \mathfrak{q} . In total, we computed about 34M relations.

The main parameters in the sieve were the following: we sieved all primes below 80M on the f -side, and below 120M on the g -side, and we allowed two large primes less than 2^{29} on each side. The search space for each special- \mathfrak{q} was set to $2^{15} \times 2^{14}$ (the parameter I in CADO-NFS was set to 15).

The total CPU time for this relation collection step is equivalent to 157 days on one core of an Intel Xeon E5-2650 at 2 GHz. This was run in parallel on a few nodes, each with 16 cores, so that the elapsed time for this step was a few days, and could easily be made arbitrary small with enough nodes.

Table 4. Comparison of running time for integer factorization (NFS-IF), discrete logarithm in prime field (NFS-DL(p)) and in quadratic field (NFS-DL(p^2)) of same global size 180 dd

Algorithm	relation collection	linear algebra	total
NFS-IF	5 years	5.5 months	5.5 years
NFS-DL(p)	50 years	80 years	130 years
NFS-DL(p^2)	157 days	18 days (GPU)	0.5 years

6.3 Linear Algebra

The filtering step was run as usual, but we modified it to take into account the Galois action on the ideals: we selected a representative ideal in each orbit under the action $x \mapsto 1/x$, and rewrote all the relations in terms of these representatives only. Indeed, it can be shown that the corresponding virtual logarithms are opposite modulo ℓ ; this amounts just to keep track of sign-change, that has to be reminded when combining two relations during the filtering, and when preparing the sparse matrix for the sparse linear algebra step. Since we keep only half of the columns in the matrix, and assuming a quadratic cost for the linear algebra step, the $x \mapsto 1/x$ automorphism saves a factor of 4, as claimed. The output of the filtering step was a matrix with about 2.7M rows and columns, having on average 86 non-zero entries per row.

Thanks to our choice of f and g , it was not necessary to add columns with Schirokauer maps. We used Jeljeli’s implementation of Block Wiedemann’s algorithm for GPUs [13, 14]. We used two sequences in parallel, on two independent NVidia GTX 680 graphic cards. The total running time for this step is equivalent to around 18.2 days on a single NVidia GTX 680 graphic card.

At the end of the linear algebra we know the virtual logarithms of almost all prime ideals of degree one above primes of at most 28 bits, and of some of those above primes of 29 bits. At this point we could test that the logs on the f -side were correct.

6.4 Computing Individual Logarithms

The last step is that of computing some individual logarithms. We used $G = t+2$ as a generator for \mathbb{F}_{p^2} and the following “random” element:

$$s = \lfloor (\pi(2^{298})/8) \rfloor t + \lfloor (\gamma \cdot 2^{298}) \rfloor.$$

We started by looking for an integer e such that $z = s^e$, seen as an element of the number field of f , is smooth. After a few dozen of core-hours, we found a value of e such that $z = z_1/z_2$ with z_1 and z_2 splitting completely into prime ideals of at most 65 bits. With the lattice-sieving software of CADO-NFS, we then performed a “special-q descent” for each of these prime ideals. The total time for descending all the prime ideals was a few minutes. Finally, we found

$$\log_G s \equiv 2762142436179128043003373492683066054037581738194144186101 \setminus \\ 9832278568318885392430499058012 \bmod \ell.$$

7 Conclusions

The present article contains new estimates for the complexity of solving DLP over non-prime finite fields. We have discovered several places in the $(\log p, n)$ plane where more methods battle to be the best ones. We have also analyzed the complexity of sieving on a domain of non-linear polynomials, and this shows the way for more algorithmic problems, so that this could be a routine problem for subsequent records.

From a practical point of view, we have demonstrated that a clever use of algebraic properties of fields occurring in DLP computations, such as finding polynomials defining number fields with automorphisms and/or Galois properties, gives a significant practical speed-up. This study will be continued elsewhere.

We gather some figures for the factorization of an 180 decimal digit composite number; the time needed for solving DLP on \mathbb{F}_p with p of 180 decimal digits taken from [6] and our computations for \mathbb{F}_{p^2} with p of 90 decimal digits.

Considering the relation collection phase only, we see that for the same object size, a DLP over \mathbb{F}_{p^2} is much easier than the corresponding factoring of an integer. This tends to contradict the usual rule-of-thumb: *The discrete logarithm problem in large characteristic finite fields is at least as hard as factoring an integer of the same size as the cardinality of the finite field.*

References

1. Bai, S., Filbois, A., Gaudry, P., Kruppa, A., Morain, F., Thomé, E., Zimmermann, P., et al.: Crible algébrique: Distribution, optimisation - NFS (2009). downloadable at <http://cado-nfs.gforge.inria.fr/>
2. Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 1–16. Springer, Heidelberg (2014)
3. Barbulescu, R., Pierrot, C.: The multiple number field sieve for medium- and high-characteristic finite fields. LMS Journal of Computation and Mathematics 17, 230–246 (2014). http://journals.cambridge.org/article_S1461157014000369
4. Barbulescu, R.: Algorithmes de logarithmes discrets dans les corps finis. Ph.D. thesis, Université de Lorraine (2013)
5. Barbulescu, R., Gaudry, P., Guillevic, A., Morain, F.: Improvements to the number field sieve for non-prime finite fields. preprint available at <http://hal.inria.fr/hal-01052449>
6. Bouvier, C., Gaudry, P., Imbert, L., Jeljeli, H., Thomé, E.: Discrete logarithms in GF(p) – 180 digits (2014), announcement available at the NMBRTHRY archives, item 004703
7. Canfield, E.R., Erdős, P., Pomerance, C.: On a problem of Oppenheim concerning “factorisatio numerorum”. J. Number Theory **17**(1), 1–28 (1983)

8. Collins, G.E., Encarnación, M.J.: Efficient rational number reconstruction. *Journal of Symbolic Computation* **20**(3), 287–297 (1995)
9. Coppersmith, D.: Modifications to the number field sieve. *J. of Cryptology* **6**(3), 169–180 (1993)
10. Coppersmith, D.: Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Math. Comp.* **62**(205), 333–350 (1994)
11. Foster, K.: HT90 and “simplest” number fields. *Illinois J. Math.* **55**(4), 1621–1655 (2011)
12. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. *J. of Cryptology* **23**(2), 224–280 (2010)
13. Jeljeli, H.: Accelerating iterative SpMV for discrete logarithm problem using GPUs (2014). <http://hal.inria.fr/hal-00734975/>, preprint, to appear in WAIFI 2014
14. Jeljeli, H.: An implementation of the Block-Wiedemann algorithm on NVIDIA-GPUs using the Residue Number System (RNS) arithmetic (2014). available from <http://www.loria.fr/~hjeljeli/>
15. Joux, A., Lercier, R.: Improvements to the general number field for discrete logarithms in prime fields. *Math. Comp.* **72**(242), 953–967 (2003)
16. Joux, A., Lercier, R., Smart, N.P., Vercauteren, F.: The Number Field Sieve in the Medium Prime Case. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 326–344. Springer, Heidelberg (2006)
17. Joux, A., Lercier, R., et al.: Algorithmes pour résoudre le problème du logarithme discret dans les corps finis. *Nouvelles Méthodes Mathématiques en Cryptographie*, volume Fascicule Journées Annuelles, p. 23 (2007)
18. Joux, A., Pierrot, C.: The Special Number Field Sieve in \mathbb{F}_{p^n} . In: Cao, Z., Zhang, F. (eds.) Pairing 2013. LNCS, vol. 8365, pp. 45–61. Springer, Heidelberg (2014)
19. Kalkbrener, M.: An upper bound on the number of monomials in determinants of sparse matrices with symbolic entries. *Mathematica Pannonica* **73**, 82 (1997)
20. Kleinjung, T.: On polynomial selection for the general number field sieve. *Mathematics of Computation* **75**(256), 2037–2047 (2006)
21. Lenstra, A.K., Verheul, E.R.: The XTR Public Key System. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 1–19. Springer, Heidelberg (2000)
22. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. *Mathematische Annalen* **261**(4), 515–534 (1982)
23. Matyukhin, D.V.: On asymptotic complexity of computing discrete logarithms over GF(p). *Discrete Mathematics and Applications* **13**(1), 27–50 (2003)
24. Matyukhin, D.: Effective version of the number field sieve for discrete logarithms in the field $GF(p^k)$. *Trudy po Discretnoi Matematike* **9**, 121–151 (2006) (in Russian). http://m.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=tdm&paperid=144&option_lang=eng
25. Murphy, B.A.: Polynomial selection for the number field sieve integer factorisation algorithm. Ph.D. thesis, Australian National Univers (1999)
26. Pierrot, C.: The multiple number field sieve with conjugation method (August 2014). preprint available at <https://eprint.iacr.org/2014/641>
27. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms over GF(p) and his cryptographic significance. *IEEE Trans. Inform. Theory* **24**(1), 106–110 (1978)
28. Pollard, J.M.: Monte Carlo methods for index computation (mod p). *Math. Comp.* **32**(143), 918–924 (1978)
29. Rubin, K., Silverberg, A.: Torus-Based Cryptography. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 349–365. Springer, Heidelberg (2003)

30. Schirokauer, O.: Using number fields to compute logarithms in finite fields. *Math. Comp.* **69**(231), 1267–1283 (2000)
31. Schirokauer, O.: Virtual logarithms. *J. Algorithms* **57**, 140–147 (2005)
32. Smith, P., Skinner, C.: A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms. In: Pieprzyk, J., Safavi-Naini, R. (eds.) *Advances in Cryptology - ASIACRYPT 1994*. LNCS, vol. 917, pp. 357–364. Springer, Heidelberg (1994)
33. Wiedemann, D.: Solving sparse linear equations over finite fields. *IEEE Trans. Inform. Theory* **32**(1), 54–62 (1986)

The Multiple Number Field Sieve with Conjugation and Generalized Joux-Lercier Methods

Cécile Pierrot^{1,2(✉)}

¹ CNRS and Direction Générale de l’Armement, Rennes, France

² Laboratoire d’Informatique de Paris 6, UPMC/Sorbonnes-Universités,
Paris, France
`Cecile.Pierrot@lip6.fr`

Abstract. In this paper, we propose two variants of the Number Field Sieve (NFS) to compute discrete logarithms in medium characteristic finite fields. We consider algorithms that combine two ideas, namely the Multiple variant of the Number Field Sieve (MNFS) taking advantage of a large number of number fields in the sieving phase, and two recent polynomial selections for the classical Number Field Sieve. Combining MNFS with the Conjugation Method, we design the best asymptotic algorithm to compute discrete logarithms in the medium characteristic case. The asymptotic complexity of our improved algorithm is $L_{p^n}(1/3, (8(9 + 4\sqrt{6})/15)^{1/3}) \approx L_{p^n}(1/3, 2.156)$, where \mathbb{F}_{p^n} is the target finite field. This has to be compared with the complexity of the previous state-of-the-art algorithm for medium characteristic finite fields, NFS with Conjugation Method, that has a complexity of approximately $L_{p^n}(1/3, 2.201)$. Similarly, combining MNFS with the Generalized Joux-Lercier method leads to an improvement on the asymptotic complexities in the boundary case between medium and high characteristic finite fields.

1 Introduction

Public key cryptosystems are designed around computational hardness assumptions related to mathematical properties, making such protocols hard to break in practice by any adversary. Algorithmic number theory provides most of those assumptions, such as the presumed difficulty to factorize a large integer or to compute discrete logarithms in some groups. Given an arbitrary element h of a cyclic group, the discrete logarithm problem consists in recovering the exponent x of a generator g such that $g^x = h$. We focus here on the multiplicative group of the invertible elements in a finite field.

Current discrete logarithms algorithms for finite fields vary with the relative sizes of the characteristic p and the extension degree n . To be more precise, finite fields split into three families and so do the related algorithms. When p is small compared to n , the best choice is to apply the recent Quasi-Polynomial algorithm [BGJT14]. Medium and high characteristics share some properties since we use in both cases variants of the Number Field Sieve (NFS) that was

first introduced for discrete logarithms computations in prime fields in 1993 by Gordon [Gor93]. Then, NFS was extended to all medium and high characteristic finite fields in 2006 by Joux, Lercier, Smart and Vercauteren [JLSV06]. For the past few months, discrete logarithm in finite fields has been a vivid domain and things change fast – not only for small characteristic.

In February 2014, Barbulescu and Pierrot [BP14] presented the Multiple Number Field Sieve (MNFS) that applies in both medium and high characteristic finite fields. As for NFS, the main idea came from factoring [Cop93] and was first introduced for discrete logarithms computations in prime fields in 2003 thanks to Matyukhin [Mat03]. In both medium and high characteristic cases, the idea is to go from two number fields, as in the classical NFS, to a large number of number fields, making the probability to obtain a good relation in the sieving phase higher. Yet, the sieving phase differs between medium and high characteristics since the parameters of the two first polynomials defining the number fields are equal in the medium case but unbalanced in the high case. Let us recall the notation $L_q(\alpha, c) = \exp((c+o(1))(\log q)^\alpha(\log \log q)^{1-\alpha})$ to be more precise about complexities, and focus on the high characteristic case. Due to unbalanced degree of the first two polynomials, the variant proposed by Barbulescu and Pierrot is dissymmetric. It means that in the sieving phase they select only elements that are small in some sense in the first number field and in at least another number field, giving to the first number field a specific role with regards to the others. With this dissymmetric MNFS, the asymptotic complexity to compute discrete logarithms in a finite field \mathbb{F}_{p^n} of characteristic $p = L_{p^n}(l_p, c)$ when p is high, *i.e.* when $l_p > 2/3$, is the same as the complexity given for factoring an integer of the same size [Cop93]. Namely, it is:

$$L_{p^n} \left(\frac{1}{3}, \left(\frac{2 \cdot (46 + 13\sqrt{13})}{27} \right)^{1/3} \right).$$

Note that MNFS as described in [BP14] is currently the state-of-the-art algorithm for computing discrete logarithms in high characteristic finite fields.

In the medium characteristic case, *i.e.* when $1/3 \leq l_p \leq 2/3$, the polynomial selection of the classical Number Field Sieve allows to construct two polynomials with same degrees and same sizes of coefficients. Making linear combination, MNFS creates then a lots of polynomials with equal parameters. Thanks to this notion of symmetry, the sieving phase of the Multiple variant consists in keeping elements that are small in any pairs of number fields, making the probability to obtain a good relation growing further.

Yet, few months later, in August 2014, Barbulescu, Gaudry, Guillevic and Morain detailed in a preprint [BGGM14] some practical improvements for the classical Number Field Sieve. Besides, they gave a new polynomial selection method that has the nice theoretical interest to lead to the best asymptotic heuristic complexity known in the medium characteristic case, overpassing the one given in [BP14]. This new polynomial selection also called Conjugation Method permits to create one polynomial with a *small* degree and *high*

coefficients and another one with a *high* degree and coefficients of constant size. Finally, the authors of [BGGM14] obtain the asymptotic complexity:

$$L_{p^n} \left(\frac{1}{3}, \left(\frac{96}{9} \right)^{1/3} \right).$$

In this article, we adapt for the first time the Multiple variant of NFS to this very recent algorithm. At first sight, one could fear that the parameters of the two polynomials given with the Conjugation Method could act as a barrier, since their unbalanced features differ from the ones used in the medium characteristic case of [BP14]. Moreover, following the high characteristic dissymmetric sieving phase of [BP14] and creating the remaining polynomials with linear combination would mean spreading both *high* coefficients and *high* degrees on the polynomials defining the various number fields. This clearly would not be a good idea, as all NFS-based algorithms require to create elements with small norms. However, we show that the Conjugation Method may be adapted to overcome this difficulty. The idea is to try to keep the advantage of the kind of *balanced dissymmetry* brought by the two polynomials with *small-degree-high-coefficients/high-degree-small-coefficients*. We show that the Multiple Number Field Sieve with Conjugation Method (MNFS-CM) becomes the best current algorithm to compute discrete logarithms in medium characteristic finite fields. Indeed, in this case its asymptotic complexity is:

$$L_{p^n} \left(\frac{1}{3}, \left(\frac{8 \cdot (9 + 4\sqrt{6})}{15} \right)^{1/3} \right).$$

To ease the comparison, note that our second constant $(8(9 + 4\sqrt{6})/15)^{1/3} \approx 2.156$ whereas the previous one is $(96/9)^{1/3} \approx 2.201$. MNFS-CM in the boundary case between medium and high characteristic leads also to an improvement of NFS-CM. Interestingly enough, sieving on degree one polynomials with MNFS-CM in this boundary case permits to obtain the best asymptotic complexity ever of any medium, boundary and high characteristic discrete logarithms algorithms, which is approximately $L_{p^n}(1/3, 1.659)$.

Besides the new Conjugation Method, the authors of [BGGM14] extend the polynomial selection given by Joux and Lercier in [JL03] for prime fields. Thanks to it, they get an improvement on the high cases of the boundary case. We propose here a simple dissymmetric Multiple Number Field Sieve based on this Generalized Joux-Lercier method (MNFS-GJL) to get a further improvement on the same boundary case. Note that the asymptotic complexity we obtain here,

$$L_{p^n} \left(\frac{1}{3}, \left(\frac{2 \cdot (46 + 13\sqrt{13})}{27} \right)^{1/3} \right),$$

is exactly the one of MNFS for high characteristic finite fields, as given in [BP14].

Outline. We first detail in Section 2 how to manage the selection of numerous polynomials based on the Conjugation method to construct a dissymmetric Multiple Number Field Sieve. Section 3 explains then how to combine MNFS with the Generalized Joux-Lercier method. The asymptotic complexity analyses of both medium and boundary cases are given in Section 4.

2 Combining the Multiple Variant of the Number Field Sieve with the Conjugation Method

Let \mathbb{F}_{p^n} denote the finite field we target, p its characteristic and n the extension degree relatively to the base field. We propose an algorithm to compute discrete logarithms in \mathbb{F}_{p^n} as soon as p can be written as $p = L_{p^n}(l_p, c_p)$ with $1/3 \leq l_p \leq 2/3$ (and c_p close to 1). In this case we say that the characteristic has medium size. In Section 2.1 we explain how to represent the finite field and to construct the polynomials that define the large number of number fields we need. In Section 2.2 we give details about the variant of the Multiple Number Field Sieve we propose to follow.

2.1 Polynomial Selection

Basic Idea: Large Numbers of Polynomials with a Common Root in \mathbb{F}_{p^n} . To compute discrete logarithms in \mathbb{F}_{p^n} , all algorithms based on the Number Field Sieve start by choosing two polynomials f_1 and f_2 with integers coefficients such that the greatest common divisor of these polynomials has an irreducible factor of degree n over the base field. If m denotes a common root of these two polynomials in \mathbb{F}_{p^n} and $\mathbb{Q}(\theta_i)$ denotes the number field $\mathbb{Q}[X]/(f_i(X))$ for each $i = 1, 2$, i.e. θ_i is a root of f_i in \mathbb{C} , then we are able to draw the commutative diagram of Figure 1.

Since MNFS requires to have a large number of number fields, let say V number fields, then we have to construct $V - 2$ extra polynomials that share the same common root m in \mathbb{F}_{p^n} . The commutative diagram that is the cornerstone of all Multiple variants of the Number Field Sieve is given in Figure 2.

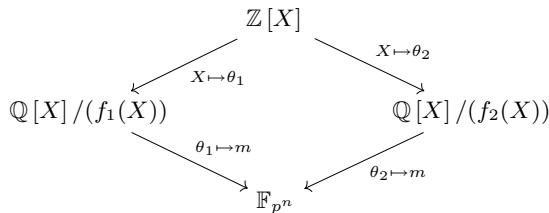


Fig. 1. Commutative diagram of NFS

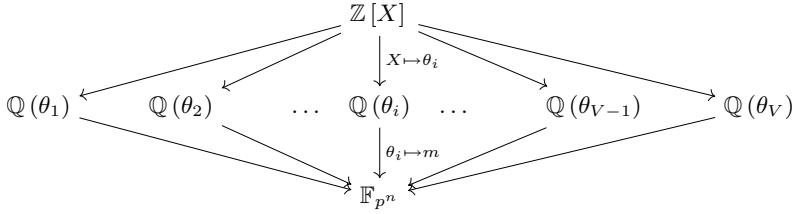


Fig. 2. Commutative diagram of MNFS

Settings: Construction of V Polynomials with the Conjugation Method. We start with the Conjugation Method given in [BGGM14, Paragraph6.3] to construct the first two polynomials. The idea is as follows.

We create two auxilliary polynomials g_a and g_b in $\mathbb{Z}[X]$ with small coefficients such that $\deg g_a = n$ and $\deg g_b < n$. We then search for an irreducible polynomial $X^2 + uX + v$ over $\mathbb{Z}[X]$, where u and v are small integers¹ of size $O(\log p)$, such that its roots λ and λ' are in \mathbb{F}_p . Since we seek a degree n irreducible polynomial over $\mathbb{F}_p[X]$ to construct the finite field, we keep the polynomial $X^2 + uX + v$ if one of the two degree n polynomials $g_a + \lambda g_b$ or $g_a + \lambda' g_b$ is irreducible over $\mathbb{F}_p[X]$. In the sequel we assume that $g_a + \lambda g_b$ is irreducible over $\mathbb{F}_p[X]$. When we have found such parameters, we set our first polynomial $f_1 \in \mathbb{Z}[X]$:

$$f_1 = g_a^2 - ug_ag_b + vg_b^2.$$

Equivalently, f_1 is defined in [BGGM14] as equal to $\text{Res}_Y(Y^2 + uY + v, g_a(X) + Yg_b(X))$. Since λ and λ' are roots of $X^2 + uX + v$ in \mathbb{F}_p , we have the equality of polynomials $f_1 \equiv g_a^2 + (\lambda + \lambda')g_ag_b + \lambda\lambda'g_b^2 \pmod{p}$. In other words, $f_1 \equiv (g_a + \lambda g_b)(g_a + \lambda' g_b) \pmod{p}$. Thus we have a polynomial f_1 of degree $2n$ with coefficients of size $O(\log p)$ that is divisible by $g_a + \lambda g_b$ in $\mathbb{F}_p[X]$.

Let us construct the next two polynomials. Thanks to continued fractions we can write:

$$\lambda \equiv \frac{a}{b} \equiv \frac{a'}{b'} \pmod{p}$$

where a, b, a' and b' are of the size of \sqrt{p} . We underline that these two reconstructions (a, b) and (a', b') of λ are linearly independent over \mathbb{Q} . We then set:

$$f_2 = bg_a + ag_b \quad \text{and} \quad f_3 = b'g_a + a'g_b.$$

¹ We correct here a mistake in [BGGM14, Paragraph6.3]. The authors propose to search for an irreducible quadratic polynomial that has constant size coefficients. However, if $|u|$ and $|v|$ are both lower than a constant C , then there exist 2^{4C^2} such polynomials. Since each one has probability $1/2$ to has its roots in \mathbb{F}_p for one random prime p , if we try to select such polynomials for approximately 2^{4C^2} primes, we will find one finite field \mathbb{F}_p for which this method fails. Looking for quadratic polynomials with coefficients of size $O(\log p)$ bypasses this trap and does not interfere with final asymptotic complexities.

Note that the Conjugation Method ends with the selection of f_1 and f_2 and does not use the second reconstruction. It is clear that both f_2 and f_3 have degree n and coefficients of size \sqrt{p} . Furthermore, we notice that $f_2 \equiv b(g_a + \lambda g_b) \pmod{p}$ and similarly $f_3 \equiv b'(g_a + \lambda g_b) \pmod{p}$, so they share a common root with f_1 in \mathbb{F}_{p^n} .

We finally set for all i from 4 to V :

$$f_i = \alpha_i f_2 + \beta_i f_3$$

with α_i and β_i of the size of \sqrt{V} . We underline that V is negligible with regards to p , as shown in Section 4. Thanks to linear combination, for all $2 \leq i \leq V$, f_i has degree n , coefficients of size \sqrt{p} and is divisible by $g_a + \lambda g_b$ in $\mathbb{F}_p[X]$.

2.2 A Dissymmetric Multiple Number Field Sieve

As any Index Calculus algorithm, the variant we propose follows three phases: the sieving phase, in which we create lots of relations involving only a small set of elements, the factor base ; the linear algebra, to recover the discrete logarithms of the elements of the factor base ; and the individual logarithm phase, to compute the discrete logarithm of an arbitrary element of the finite field.

We propose to sieve as usual on high degree polynomials $\phi(X) = a_0 + \cdots + a_{t-1}X^{t-1}$ with coefficients of size bounded by S . Let us recall that, given an integer y , an integer x is called *y-smooth* if it can be written as a product of prime factors less than y . We then collect all polynomials such that, first, the norm of $\phi(\theta_1)$ is B -smooth and, second, there exists (at least) one number field $\mathbb{Q}(\theta_i)$ with $i \geq 2$ in which the norm $\phi(\theta_i)$ is B' -smooth. In other simpler words, we create relations thanks to polynomials that cross over the diagram of Figure 3 in two paths: the one on the left side of the drawing and (at least) another one among those on the right. If we set that the factor base consists in the union of all the prime ideals in the rings of integers that have a B -or- B' -smooth norm, the smoothness bound depending on the number field, then we keep only relations that involve these factor base elements. Note that B and B' are two smoothness bounds possibly different from one another.

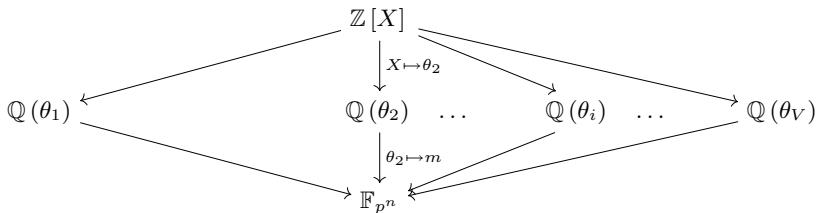


Fig. 3. Commutative diagram for the dissymmetric Multiple Number Field Sieve with Conjugation Method

After the same post-processing as in [JLSV06] or as detailed in [BGGM14] more recently, each such polynomial ϕ yields a linear equation between “logarithms of ideals” coming from two number fields. Hence, from each relation we obtain a linear equation where the unknowns are the logarithms of ideals. Let us remark that by construction each equation only involves a small number of unknowns.

The sparse linear algebra and individual logarithm phases run exactly as in the classical Number Field Sieve of [JLSV06]. Even if there exists a specific way to manage the last phase with a multiple variant as detailed in [BP14], taking advantage of the large number of number fields again, we do not consider it here. In fact, the runtime of the classical individual logarithm phase is already negligible with regards to the total runtime of the algorithm, as proved by Barbulescu and Pierrot in their article.

3 Combining the Multiple Number Field Sieve with the General Joux-Lercier Method

In 2003 Joux and Lercier [JL03] gave a polynomial selection to compute discrete logarithms in prime fields. Barbulescu, Gaudry, Guillaud and Morain propose in [BGGM14, Paragraph 6.2] to generalize this construction. Using again lattice reduction, they obtain an improvement on the asymptotic complexity in the boundary case where the characteristic can be written as $p = L_Q(2/3, c)$ for some specific c . We propose here to apply a Multiple variant of NFS to this construction in a very simple way.

Let us recall the General Joux-Lercier (GJL) method as presented in [BGGM14]. In order to compute discrete logarithms in the finite field \mathbb{F}_{p^n} , we first select an irreducible polynomial f_1 in $\mathbb{F}_p[X]$ with small coefficients (let us say of the size of $O(\log p^n)$) and such that it has an irreducible factor φ of degree n modulo p . We assume furthermore that this irreducible factor is monic. Let us write $\varphi = X^n + \sum_{i=0}^{n-1} \varphi_i X^i$ and $d+1$ the degree of f . Thus we have $d+1 > n$.² To assure that the second polynomial shares the same irreducible factor modulo p , we define it thanks to linear combination of polynomials of the form φX^k and pX^k . Lattice reduction permits then to obtain small coefficients. More precisely, we note M the following $(d+1) \times (d+1)$ matrix:

$$M = \left(\begin{array}{cccccc|c} & & & & 1 & & X^d \\ & & & & \ddots & \varphi_{n-1} & X^{d-1} \\ & & & 1 & \ddots & \vdots & \vdots \\ p & & \varphi_{n-1} & \ddots & \varphi_0 & & X^{n-1} \\ \underbrace{\dots}_{n \text{ columns}} & \underbrace{\vdots}_{\varphi_0} & \ddots & & & & \vdots \\ p & & & & & & 1 \end{array} \right)$$

² We emphasize that we require φ to be different from f_1 since we need that f_2 is not equal to $f_1 \bmod p$.

A generator of this lattice of polynomials is represented in one column, meaning that each one of its coefficients is written in the row corresponding to the associated monomial (see indications on the right of the matrix). Clearly, the determinant of the lattice is p^n and its dimension is $d+1$. Hence, running the LLL algorithm on M gives a polynomial of degree at most d that has coefficients of size at most $p^{n/d+1}$ (assuming that $2^{(d+1)/4}$ stays small compared to $p^{n/d+1}$).

In a nutshell, we obtain two polynomials f_1 and f_2 that share a common degree n factor over $\mathbb{F}_p[X]$ and such that:

$$\begin{aligned}\deg f_1 &= d+1 > n, \|f_1\|_\infty = O(\log p^n), \\ \deg f_2 &= d, \|f_2\|_\infty = p^{n/(d+1)}.\end{aligned}$$

where $\|f_i\|_\infty$ denotes the largest coefficients of f_i in absolute value. This ends the GJL method. As in [BP14], we perform then linear combination of these two polynomials. Setting for all i from 3 to V :

$$f_i = \alpha_i f_2 + \beta_i f_3$$

with α_i and β_i of the size of \sqrt{V} . Thus, for all $3 \leq i \leq V$, f_i has degree $d+1$ and coefficients of size $p^{n/(d+1)}$. Note that it is also possible to extract from the lattice reduction a second polynomial f_3 that has, as f_2 , degree d and coefficients of size $p^{n/(d+1)}$. Making linear combination of f_2 and f_3 leads to polynomials of degree d instead of degree $d+1$. Yet, this little improvement has no impact on the asymptotic complexity of the algorithm.

As usual in this boundary case where $p = L_Q(2/3, c)$, we propose to sieve on degree 1 polynomials. We apply then a dissymmetric MNFS, as described in Section 2.2.

4 Asymptotic Complexity Analyses

We give now details about the asymptotic heuristic complexities we obtain with MNFS-CM in medium characteristic and with both MNFS-CM and MNFS-GJL in the boundary case between medium and high characteristics. Let us fix the notations. We write the extension degree n and the characteristic p of the target finite field \mathbb{F}_Q as:

$$n = \frac{1}{c_p} \left(\frac{\log Q}{\log \log Q} \right)^{1-l_p} \quad \text{and} \quad p = \exp(c_p (\log Q)^{l_p} (\log \log Q)^{1-l_p})$$

with $1/3 \leq l_p \leq 2/3$. The parameters taking part in the heuristic asymptotic complexity analyses are: the sieving bound S , the degree of the polynomials we are sieving over $t-1$, the number of number fields V , the smoothness bound B related to the first number field and the smoothness bound B' related to the others number fields. The analyses of both MNFS-CM and MNFS-GJL work by optimizing the total runtime of the sieving and linear algebra phases while complying with two constraints.

Balancing the Cost of the Two First Phases. We first require that the runtime of the sieving phase S^t equals the cost of the linear algebra. Since the linear system of equations we obtain is sparse, the cost of the linear algebra is asymptotically $(B + VB')^2$. Similarly to balancing the runtime of the two phases, we require that $B = VB'$. Thus, leaving apart the constant 4 that is clearly negligible with regards to the sizes of the parameters, the first constraint can be written as:

$$S^t = B^2. \quad (1)$$

Balancing the Number of Equations with the Number of Unknowns. To be able to do the linear algebra phase correctly, we require that the number of unknowns, that is approximately B , is equal to the number of equations produced in the sieving phase. If we note \mathcal{P} the probability that a polynomial give a good relation then we want to have $S^t \mathcal{P} = B$. Combining it with the constraint (1), it leads to:

$$B = 1/\mathcal{P}.$$

4.1 Analysis of MNFS-CM in the Medium Characteristic Case

We continue the analysis for the large range of finite fields where the characteristic can be written as $p = L_Q(l_p, c_p)$ with $1/3 \leq l_p < 2/3$. We consider here MNFS-CM as described in Section 2.

Evaluating the Probability of Smoothness. To evaluate the probability \mathcal{P} we need to recall some tools about norms in number fields. For $f_i \in \mathbb{Z}[X]$ an irreducible polynomial, θ_i a complex root of f_i , and for any polynomial $\phi \in \mathbb{Z}[X]$, the norm $N(\phi(\theta))$ satisfies $\text{Res}(\phi, f_i) = \pm l_i^{\deg \phi} N(\phi(\theta))$, where the term l_i is the leading coefficient of f_i . Since we treat l_i together with small primes, we make no distinction in smoothness estimates between norms and resultants. We have the upper bound on the resultant:

$$|\text{Res}(\phi, f_i)| \leq (\deg f_i + \deg \phi)! \cdot \|f_i\|_{\infty}^{\deg \phi} \cdot \|\phi\|_{\infty}^{\deg f_i}.$$

Thus, recalling that f_1 is of degree $2n$ and has constant coefficients and that every other polynomials f_i has degree n and coefficients of the size \sqrt{p} , we obtain that the norm of a sieving polynomial ϕ is upper-bounded by S^{2n} in the first number field and by $S^n p^{t/2}$ in every other number fields. To evaluate the probability of smoothness of these norms with regards to B and B' , the main tool is the following theorem:

Theorem 1 (Canfield, Erdős, Pomerance [CEP83]). *Let $\psi(x, y)$ denote the number of positive integers up to x which are y -smooth. If $\epsilon > 0$ and $3 \leq u \leq (1 - \epsilon) \log x / \log \log x$, then $\psi(x, x^{1/u}) = xu^{-u+o(u)}$.*

Yet, this result under this form is not very convenient. If we write the two integers x and y with the L_q -notation, we obtain a more helpful corollary:

Corollary 1. Let $(\alpha_1, \alpha_2, c_1, c_2) \in [0, 1]^2 \times [0, \infty)^2$ be four reals such that $\alpha_1 > \alpha_2$. Let \mathcal{P} denote the probability that a random positive integer below $x = L_q(\alpha_1, c_1)$ splits into primes less than $y = L_q(\alpha_2, c_2)$. Then we have $\mathcal{P}^{-1} = L_q(\alpha_1 - \alpha_2, (\alpha_1 - \alpha_2)c_1c_2^{-1})$.

So we would like to express both norms and sieving bounds with the help of this notation. As usual, we set:

$$t = \frac{c_t}{c_p} \left(\frac{\log Q}{\log \log Q} \right)^{2/3-l_p}, \quad S^t = L_Q(1/3, c_s c_t), \quad B = L_Q(1/3, c_b) \quad \text{and} \\ V = L_Q(1/3, c_v).$$

Thanks to this, we first remark that the first constraint can be rewritten as:

$$c_s c_t = 2c_b. \quad (2)$$

Besides, we apply the Corollary 1 to reformulate the second constraint. Let us note $L_Q(1/3, p_r)$ (respectively $L_Q(1/3, p_{r'})$) the probability to get a B -smooth norm in the first number field (respectively a B' -smooth norm in at least one other number field). The second constraint becomes $c_b = -(p_r + p_{r'})$. Using equation (2), the constants in the probabilities can be written as:

$$p_r = \frac{-2c_s}{3c_b} = \frac{-2(2/c_t)c_b}{3c_b} \quad \text{and} \quad p_{r'} = c_v - \frac{(2/c_t)c_b + c_t/2}{3(c_b - c_v)}.$$

That leads to require $c_b = -(-4/(3c_t) + c_v - (4c_b + c_t^2)/(6c_t(c_b - c_v)))$ and afterwards $6c_t(c_b^2 - c_v^2) = 8(c_b - c_v) + 4c_b + c_t^2$. Finally we would like to have:

$$(6c_t)c_b^2 - 12c_b - 6c_t c_v^2 + 8c_v - c_t^2 = 0. \quad (3)$$

Optimizing the Asymptotic Complexity. We recall that the complexity of our algorithm is given by the cost of the sparse linear algebra $L_Q(1/3, 2c_b)$, since we equalize the runtime of the sieving and linear algebra phases. Hence we look for minimizing c_b under the above constraint (3). The method of Lagrange multipliers indicates that c_b, c_v and c_t have to be solutions of the following system:

$$\begin{cases} 2 + \lambda(12c_t c_b - 12) = 0 \\ \lambda(-12c_v c_t + 8) = 0 \\ \lambda(6c_b^2 - 6c_v^2 - 2c_t) = 0 \end{cases}$$

with $\lambda \in \mathbb{R}^*$. From the second row we obtain $c_t = 2/(3c_v)$ and from the third one we get $c_b = (c_v^2 + 2/(9c_v))^{1/2}$. Together with equation (3), it gives the equation in one variable: $405c_v^6 + 126c_v^3 - 1 = 0$. We deduce that $c_v = ((3\sqrt{6} - 7)/45)^{1/3}$ and we recover $c_b = ((9 + 4\sqrt{6})/15)^{1/3}$. Finally, the heuristic asymptotic complexity of the Multiple Number Field Sieve with Conjugation Method is, as announced:

$$L_Q \left(\frac{1}{3}, \left(\frac{8 \cdot (9 + 4\sqrt{6})}{15} \right)^{1/3} \right).$$

This has to be compared with the Number Field Sieve with Conjugation Method proposed in [BGGM14] that has complexity $L_Q(1/3, (96/9)^{1/3})$. Note that our second constant is $(8(9 + 4\sqrt{6})/15)^{1/3} \approx 2.156$, whereas $(96/9)^{1/3} \approx 2.201$.

4.2 Analysis of MNFS-CM in the Boundary Case $p = L_Q(2/3, c_p)$

The analysis made in this case follows the previous one except for the fact that we have to reconsider the parameter t . We consider here a family of algorithms indexed by the degree $t - 1$ of the polynomials of the sieving. We compute so the final complexity of each algorithm as a function of c_p (and t). Moreover, we underline that the round off error in t in the computation of the norms is no longer negligible.

Sieving on Polynomials of Degree $t - 1$. Again, to easily evaluate the probability of smoothness of norms, we set the following parameters:

$$V = L_Q(1/3, c_v), \quad B = L_Q(1/3, c_b), \quad B' = L_Q(1/3, c_b - c_v) \quad \text{and} \\ S = L_Q(1/3, c_s).$$

With these notations, the first constraint becomes this time:

$$c_s t = 2c_b. \tag{4}$$

Moreover, the norms are upper-bounded by $S^{2n} = L_Q(2/3, 2c_s/c_p)$ in the first number field and by $S^n p^{(t-1)/2} = L_Q(2/3, c_s/c_p + c_p(t-1)/2)$ in all the other number fields. We apply the Canfield-Erdős-Pomerance theorem, and, with the same notation as in the previous paragraph, we obtain $p_r = -2c_s/(3c_b c_p)$ in one hand and $p_{r'} = c_v - (c_s/c_p + c_p(t-1)/2)/(3(c_b - c_v))$ in the other hand. Using equation (4), the second constraint $c_b = -(p_r + p_{r'})$ can be rewritten as $3tc_p(c_b - c_v)(c_b + c_v) = 4(c_b - c_v) + 2c_b + t(t-1)c_p^2/2$. As a consequence, we require:

$$(6tc_p)c_b^2 - 12c_b - 6tc_p c_v^2 + 8c_v - t(t-1)c_p^2 = 0. \tag{5}$$

As previously, we want to minimize $2c_b$ under the constraint (5). The method of Lagrange multipliers shows that we need that the derivative of $(6tc_p)c_b^2 - 12c_b - 6tc_p c_v^2 + 8c_v - t(t-1)c_p^2$ with respect to c_v is equal to 0. This leads to require that $c_v = 2/(3tc_p)$. Putting this value in equation (5) we get:

$$(18t^2 c_p^2)c_b^2 - (36tc_p)c_b + 8 - 3t^2(t-1)c_p^3 = 0.$$

Finally, solving this equation in c_b we deduce that $c_b = (6 + (20 + 6t^2(t-1)c_p^3)^{1/2})/(6tc_p)$. Consequently, the asymptotic complexity of the Multiple Number Field Sieve with Conjugation Method in this boundary case is:

$$L_Q \left(\frac{1}{3}, \frac{2}{c_p t} + \sqrt{\frac{20}{(9c_p t)^2} + \frac{2}{3}c_p(t-1)} \right)$$

where $t - 1$ is the degree of the polynomials we are sieving on. Figure 4 compares our MNFS-CM with previous and various algorithms in this boundary case. For almost all variants of the Number Field Sieve presented in this figure (namely NFS, MNFS, NFS-CM and MNFS-CM), each hollow in the curve corresponds to a particular degree of the polynomials we are sieving on.

Remark 1. *This boundary case has been the scene of various recent improvements but, as far as we know, all of them are not yet published nor available on the Internet. In particular, this is the case of the so-called PiRaTh algorithm, presented at the DLP conference in May 2014 by Pierrick Gaudry, Razvan Barbulescu and Thorsten Kleinjung. Yet, for the sake of comparison, we plot it together with already broadcast algorithms.*

The Best Asymptotic Complexity of any Variant of the Number Field Sieve: MNFS-CM on Linear Polynomials. According to Figure 4, sieving on linear polynomials seems to give the best complexity, as usual in this boundary case. Let us make a more precise analysis of the optimal case reached by our Multiple Number Field Sieve with Conjugation Method. We consider now c_p as a variable and we would like to find the minimal complexity obtained by each algorithm. Namely, we want to minimize:

$$C(c_p) = \frac{2}{c_p t} + \sqrt{\frac{20}{(9c_p t)^2} + \frac{2}{3} c_p(t-1)}.$$

The derivative of this function with respect to c_p vanishes when $2 \cdot 9^2 t c_p (20/(9c_p t)^2 + (2/3)c_p(t-1))^{1/2} = -20 + 27(t-1)t^2 c_p^3$. This leads to the quadratic equation in c_p^3 : $3^6 t^4 (t-1)^2 c_p^6 - 2^4 3^3 43 t^2 (t-1) c_p^3 - 2^6 \cdot 5 \cdot 19 = 0$. Thus, the optimal value comes when $c_p = (2/3) \cdot ((43 + 18\sqrt{6})/(t^2(t-1)))^{1/3}$. We get for this value the minimal complexity:

$$L_Q \left(\frac{1}{3}, \left(\frac{9 + \sqrt{177 + 72\sqrt{6}}}{3 \cdot (43 + 18\sqrt{6})^{1/3}} \right) \cdot \left(\frac{t-1}{t} \right)^{1/3} \right).$$

Looking at this formula, it is clear that the best possible complexity is obtained when $t = 2$, *i.e.* when we sieve on linear polynomials. Interestingly enough, we conclude that we have with our MNFS-CM the best complexity of any medium, boundary and high characteristics cases, which is:

$$L_Q \left(\frac{1}{3}, \frac{9 + \sqrt{177 + 72\sqrt{6}}}{3 \cdot (2 \cdot (43 + 18\sqrt{6}))^{1/3}} \right).$$

Note that the approximation of the second constant in the complexity is given by $(9 + \sqrt{177 + 72\sqrt{6}}) \cdot 3^{-1} \cdot (2 \cdot (43 + 18\sqrt{6}))^{-1/3} \approx 1.659$. We get this complexity when p can be written as $p \approx L_Q(1/3, 1.86)$.

4.3 Analysis of MNFS-GJL in the Boundary Case $p = L_Q(2/3, c_p)$

In this setting, we recall that we propose to sieve on linear polynomials. As usual, we assume that $B = VB'$ where V is the number of number fields and B' is the smoothness bound relatively to the last $V - 1$ number fields. Thus, the constraint given in Equation (1) leads to require that the sieving bound S is equal to the first smoothness bound B . With the same notations as previously, we also require that $B = 1/\mathcal{P}$. Finally, we emphasize that the polynomial selection proposed in Section 3 requires that $n < d + 1$. If we set that:

$$d = \delta \left(\frac{\log Q}{\log \log Q} \right)^{1/3},$$

where δ is a parameter to define, then we have to keep in mind that our complexity results are valid provided $\delta \geq 1/c_p$.

Since f_1 has small coefficients and degree $d + 1$ the norms in the first number field are upper-bounded by $L_Q(2/3, c_b \delta)$. The probability to get a B -smooth norm is though $L_Q(1/3, p_r)$ with $p_r = -\delta/3$. Similarly, the norms in the last $V - 1$ number fields are bounded by $L_Q(2/3, c_b \delta + 1/\delta)$. The probability to get a B' -smooth norm in a least one number field is $L_Q(1/3, p_{r'})$ where $p_{r'} = -(c_b \delta + 1/\delta)/(c_b - c_v) + c_v$.

From $c_b = -(p_r + p_{r'})$ we get then:

$$\begin{aligned} c_b + c_v &= \frac{\delta}{3} + \frac{\delta^2 c_b + 1}{3\delta(c_b - c_v)} \\ \Leftrightarrow 3\delta(c_b^2 - c_v^2) &= 2\delta^2 c_b - \delta^2 c_v + 1 \\ \Leftrightarrow 3\delta c_b^2 - 2\delta^2 c_b + \delta^2 c_v - 3\delta c_v^2 - 1 &= 0. \end{aligned}$$

The method of Lagrange multipliers shows that we require:

$$\left\{ \begin{array}{l} 3\delta c_b^2 - 2\delta^2 c_b + \delta^2 c_v - 3\delta c_v^2 - 1 = 0 \\ 3c_b^2 - 4\delta c_b + 2\delta c_v - 3c_v^2 = 0 \\ \delta^2 - 6\delta c_v = 0 \end{array} \right. \quad (6)$$

From the third line of System (6) we recover $\delta = 6c_v$. Substituting in the second line, we obtain $c_b^2 - 8c_v c_b + 3c_v^2 = 0$. Then, writing c_v as function of c_b we get: $c_v = ((4 - \sqrt{13})/3)c_b$. Substituting the value of δ in the first line of the system gives $18c_v c_b^2 - 72c_v^2 c_b + 18c_v^3 - 1 = 0$, and, substituting again with the value of c_v we finally get: $c_b = (46 + 13\sqrt{13}/108)^{1/3}$. With this constant, we recover the value of δ which is $(4\sqrt{13} - 14)^{1/3}$. Thus, as soon as:

$$c_p \geq \left(\frac{7 + 2\sqrt{13}}{6} \right)^{1/3},$$

which is approximately equal to 1.33, the complexity of the Multiple Number Field Sieve with the Generalized Joux-Lercier method is:

$$L_Q \left(\frac{1}{3}, \left(\frac{2 \cdot (46 + 13\sqrt{13})}{27} \right)^{1/3} \right).$$

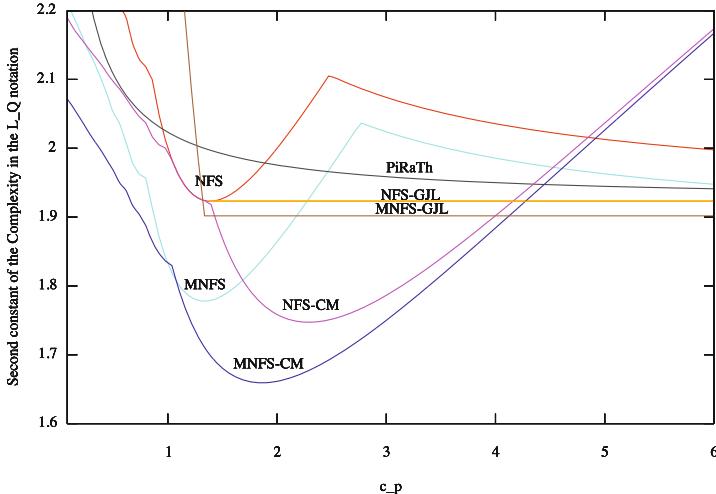


Fig. 4. Asymptotic complexities $L_Q(1/3, C(c_p))$ in the boundary case, as a function of c_p with $p = L_Q(2/3, c_p)$. The dark blue curve represents the complexities obtained with our Multiple Number Field Sieve with Conjugation Method while the brown one represents the complexity of the Multiple Number Field Sieve with the General Joux-Lercier method (see next Section). The red, light blue, black, yellow and purple curves represent respectively the complexities of NFS [JLSV06], MNFS [BP14], PiRaTh, NFS-GJL [BGGM14] and NFS-CM [BGGM14].

As expected, we recover the exact asymptotic complexity given by [BP14] when solving the discrete logarithm problem in high characteristic finite fields. This has to be compared with the asymptotic complexity of the classical Number Field Sieve with the Generalized Joux-Lercier method [BGGM14] in the same case which is $L_Q(1/3, (64/9)^{1/3})$. For the sake of comparison we recall that $(64/9)^{1/3} \approx 1.92$ whereas $(2(46 + 13\sqrt{13})/27)^{1/3} \approx 1.90$.

When $c_p < ((7 + 2\sqrt{13})/6)^{1/3}$, from the constraint $\delta > 1/c_p$ we get $\delta > (4\sqrt{13} - 14)^{1/3}$ and the previous simplification no longer applies. Yet, the equalities $c_b = 3c_v/(4 - \sqrt{13}) = \delta/(2(4 - \sqrt{13}))$ show that we minimize the complexity when $\delta = 1/c_p$. We obtain thus $c_b = (4 + \sqrt{13})/(6c_p)$. Finally, when:

$$c_p < \left(\frac{7 + 2\sqrt{13}}{6} \right)^{1/3},$$

the asymptotic complexity of MNFS with the Generalized Joux-Lercier method is:

$$L_Q \left(\frac{1}{3}, \frac{4 + \sqrt{13}}{3c_p} \right).$$

Figure 4 shows how this asymptotic complexity varies with c_p .

References

- [BGGM14] Barbulescu, R., Gaudry, P., Guillevic, A., Morain, F.: Improvements to the number field sieve for non-prime finite fields. INRIA Hal Archive, Report 01052449 (2014)
- [BGJT14] Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 1–16. Springer, Heidelberg (2014)
- [BP14] Barbulescu, R., Pierrot, C.: The multiple number field sieve for medium and high characteristic finite fields. LMS Journal of Computation and Mathematics **17**, 230–246 (2014)
- [CEP83] Canfield, E.R., Erds, P., Pomerance, C.: On a problem of Oppenheim concerning factorisatio numerorum. Journal of Number Theory **17**, 1–28 (1983)
- [Cop93] Coppersmith, D.: Modifications to the number field sieve. J. Cryptology **6**(3), 169–180 (1993)
- [Gor93] Gordon, D.M.: Discrete logarithms in GF(P) using the number field sieve. SIAM J. Discrete Math. **6**(1), 124–138 (1993)
- [JL03] Joux, A., Lercier, R.: Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the gaussian integer method. Math. Comput. **72**(242), 953–967 (2003)
- [JLSV06] Joux, A., Lercier, R., Smart, N.P., Vercauteren, F.: The Number Field Sieve in the Medium Prime Case. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 326–344. Springer, Heidelberg (2006)
- [Mat03] Matyukhin, D.V.: On asymptotic complexity of computing discrete logarithms over GF(p). Discrete Mathematics and Applications **13**(1), 27–50 (2003)

Algorithmic Cryptanalysis

Better Algorithms for LWE and LWR

Alexandre Duc^(✉), Florian Tramèr, and Serge Vaudenay

EPFL, 1015 Lausanne, Switzerland

{alexandre.duc,florian.tramer,serge.vaudenay}@epfl.ch

Abstract. The Learning With Error problem (LWE) is becoming more and more used in cryptography, for instance, in the design of some fully homomorphic encryption schemes. It is thus of primordial importance to find the best algorithms that might solve this problem so that concrete parameters can be proposed. The BKW algorithm was proposed by Blum et al. as an algorithm to solve the Learning Parity with Noise problem (LPN), a subproblem of LWE. This algorithm was then adapted to LWE by Albrecht et al.

In this paper, we improve the algorithm proposed by Albrecht et al. by using multidimensional Fourier transforms. Our algorithm is, to the best of our knowledge, the fastest LWE solving algorithm. Compared to the work of Albrecht et al. we greatly simplify the analysis, getting rid of integrals which were hard to evaluate in the final complexity. We also remove some heuristics on rounded Gaussians. Some of our results on rounded Gaussians might be of independent interest. Moreover, we also analyze algorithms solving LWE with discrete Gaussian noise.

Finally, we apply the same algorithm to the Learning With Rounding problem (LWR) for prime q , a deterministic counterpart to LWE. This problem is getting more and more attention and is used, for instance, to design pseudorandom functions. To the best of our knowledge, our algorithm is the first algorithm applied directly to LWR. Furthermore, the analysis of LWR contains some technical results of independent interest.

1 Introduction

The Learning With Error problem (LWE) was introduced by Regev in [43] and can be seen as an extension of the Learning (from) Parity with Noise problem (LPN). Roughly, the adversary is given queries from an LWE oracle, which returns uniformly random vectors \mathbf{a}_j in \mathbb{Z}_q and their inner-product with a fixed secret vector $\mathbf{s} \in \mathbb{Z}_q^k$ to which some noise was added (typically some discrete Gaussian noise). The goal of the adversary is then to recover the secret \mathbf{s} . In LPN, $q = 2$ and the noise follows a Bernoulli distribution. In his seminal paper [43], Regev shows a quantum reduction from some well-known Lattice problems like the decisional shortest vector problem (Gap-SVP) or the short independent vector problem (SIVP) to the LWE problem. Later, Peikert and Brakerski et al. showed how to make this reduction classical [16, 42]. The LWE

A. Duc—Supported by a grant of the Swiss National Science Foundation, 200021-143899/1.

problem was then used to design a wide range of cryptographic primitives. For instance, Gentry et al. showed how to construct a trapdoor function based on LWE and created an identity-based cryptosystem [26]. Applebaum et al. used LWE to design encryption schemes with strong security properties [4]. However, the biggest breakthrough that uses LWE is its use in the design of (fully) homomorphic encryption schemes (FHE). FHE was first introduced by Gentry in his PhD thesis [25]. While the initial construction was not using the LWE problem, most of the recent designs are, e.g., [15, 17, 27].

The Learning With Rounding problem (LWR) was introduced by Banerjee, Peikert, and Rosen to construct pseudorandom functions [8]. LWR can be seen as a derandomization of LWE where the random noise is replaced by a rounding modulo $p < q$. This rounding introduces a deterministic error which makes the problem hard to solve. Banerjee et al. showed that the hardness of the LWE problem can be reduced to the hardness of LWR, when $q/p = k^{\omega(1)}$, where k is the length of the secret. The LWR problem was later revisited by Alwen et al. to get rid of this exponential blowup [3]. However, the number of LWR samples given to the adversary is limited in this case. LWR finds new applications every year. Among them, there is the design of pseudorandom functions [8], lossy trapdoor functions and reusable extractors [3], or key-homomorphic PRFs [13].

When designing a new cryptosystem, one critical part is to propose some concrete parameters so that the new scheme can be used in practice. Regarding the LWE problem, there was no such algorithmic analysis before the work of Albrecht et al. [1]. This lack of concrete complexity analysis implied that most of the constructions based on LWE propose only asymptotic parameters. Hence, it is of primary importance to study algorithms that solve the hard problems on which our cryptosystems rely.

Previous Work. Algorithms solving LWE can be divided into two categories: those finding short vectors in a lattice using, e.g., Regev’s [43] or Brakerski et al.’s [16] reduction and those attacking the LWE problem directly. The first type of algorithms is extensively studied (see, e.g., [9, 21, 23, 30, 31, 36, 40, 41]). However, there is still no precise complexity analysis for large dimensions. In this paper, we focus only on the second type of algorithms the study of which started with the LPN problem and the Blum—Kalai—Wasserman algorithm (BKW) [11] with complexity $2^{O(k/\log k)}$ where k is the length of the secret vector. The idea of BKW is to add queries together, such that the vectors \mathbf{a}_j are zero in all but one positions. Then, using a majority rule, one can recover the corresponding bit of the secret with good probability.

In [35], Levieil and Fouque proposed an optimization of the BKW algorithm for LPN, denoted LF1, which recovers a full block of b bits of the secret \mathbf{s} at once by cleverly applying a Walsh-Hadamard transform. Compared to the original BKW algorithm, their method has the advantage of making use of all the available samples after reduction, instead of having to discard those with more than one non-zero position. Instead of an exhaustive search, they use a fast Walsh-Hadamard transform to recover the most likely secret block in time $O(m + b^2)$ (where m is the number of samples left after reduction). The analysis of their

algorithm shows that it clearly outperforms the standard BKW, although their asymptotic complexities remain the same. In the same paper, they also proposed to apply some heuristics to reduce the query complexity (LF2 algorithm).

In parallel, Fossorier et al. also improved the original BKW algorithm using techniques taken from fast correlation attacks [22]. Later, Bernstein and Lange [10] combined both the LF algorithms and Fossorier et al.'s work to attack Lapin [32], an authentication protocol based on a version of LPN over a ring (ring-LPN). However, all these results achieve the same asymptotic complexity of $2^{O(k/\log k)}$.

Many cryptographic applications of LPN make sure that the number of queries to the LPN oracle is limited. However, all the algorithms based on BKW initially require a sub-exponential number of LPN samples. In [37], Lyubashevsky proposed a clever way to combine queries using a universal hash function. He could, thus, obtain an algorithm using less queries (the minimal being $k^{1+2/\log k}$ for a worse time complexity).

In ICALP 2011, Arora and Ge publish the first algorithm targeting a specific version of LWE, namely when the Gaussian noise is low [5]. Using BKW for LWE was first mentioned by Regev [43]. However, it is only in 2013 that the first detailed analysis of a generic algorithm targeting LWE is published by Albrecht et al. [1]. It is an adaptation of the original BKW algorithm with some clever improvements of the memory usage and achieves complexity $2^{O(k)}$. Their analysis is extremely detailed and we present their results in Section 3. Finally, Albercht et al. presented in PKC 2014 an algorithm targeting LWE when the secret vector has small components (typically binary). Using BKW along with modulus switching techniques, they managed to reduce the complexity for solving the LWE problem in these cases [2].

Our Contribution. We contributed in the following:

- First we propose a *new algorithm for LWE*, which is better than the current state of the art. Our new algorithm replaces the log-likelihood part from [1] by a multidimensional Fourier transform. We also propose a heuristic adapted from LF2 [35] to reduce the number of oracle queries even further.
- Albrecht et al. in [1] were relying on the heuristic that the sum of rounded Gaussian variables remain rounded Gaussians. We *remove this heuristic* by a careful analysis. In particular, we give good bounds on the expected value of the cosine of the rounded Gaussian distribution. Our algorithm relies solely on the common heuristic stating that after having performed all the XORs in the BKW algorithm, all the noises are independent. This heuristic is already used in most of the LPN-solving algorithms (e.g. [1, 22, 35]).
- In [1], only the rounded Gaussian distribution for the noise in LWE is considered. While this distribution was initially used by Regev [43], more recent papers tend to use the discrete Gaussian distribution instead. We perform our analysis for both distributions.
- Albrecht et al.'s complexity is rather difficult to estimate when $\sqrt{2^a}\sigma > q/2$ [1, Theorem 2]. Indeed, their result contains a parameter which they

could express only using an integral and the erf function. Our detailed analysis allows us to bound the Fourier coefficients of the rounded Gaussian distribution in all the cases and, hence, all our complexities are simple to evaluate.

- We adapt Lyubashevsky’s idea to LWE and show that for LWE (and LWR), the minimum number of queries required using his method is $k^{1+(\log q+1)/\log k}$.
- We propose the *first algorithmic analysis of the LWR problem* when q is prime. While our proposal requires a subexponential number of samples, our detailed analysis contains many results of independent interest.

Organization. In Section 2, we introduce the LWE and LWR problems and give basic results about Gaussians and Fourier transforms. In Section 3, we present the BKW algorithm as it was done in [1]. We detail our algorithm and apply it to LWE in Section 4. We adapt it to LWR in Section 5. Finally, we conclude in Section 6.

2 Preliminaries

2.1 Notations

Given a vector \mathbf{a} we denote by \mathbf{a}_j its j -th component. We write $\mathbf{a}^{(j)}$ to say that we access the j -th vector of a set. We let $\lfloor \cdot \rfloor: \mathbb{R} \rightarrow \mathbb{Z}$ be the rounding function that rounds to the closest integer.¹ We define $\sqrt{-1} = i \in \mathbb{C}$. Finally, for a predicate $\pi(x)$, we denote by $\mathbb{1}_{\{\pi(x)\}}$ the function which is 1 when $\pi(x)$ is true and 0 otherwise.

2.2 The LWE Problem

In this section, we define the LWE problem.

Definition 1 (LWE Oracle). Let k, q be positive integers. A Learning with Error (LWE) oracle $\Pi_{\mathbf{s}, \chi}$ for a hidden vector $\mathbf{s} \in \mathbb{Z}_q^k$ and a probability distribution χ over \mathbb{Z}_q is an oracle returning

$$\left(\mathbf{a} \xleftarrow{U} \mathbb{Z}_q^k, \langle \mathbf{a}, \mathbf{s} \rangle + \nu \right),$$

where $\nu \leftarrow \chi$.

Definition 2 (Search-LWE). The Search-LWE problem is the problem of recovering the hidden secret \mathbf{s} given n queries $(\mathbf{a}^{(j)}, c^{(j)}) \in \mathbb{Z}_q^k \times \mathbb{Z}_q$ obtained from $\Pi_{\mathbf{s}, \chi}$.

In typical schemes based on LWE, the parameter q is taken to be polynomial in k , and χ follows a discretized Gaussian distribution (see next section).

¹ In case of equality, we take the floor.

2.3 Gaussian Distributions

Let $\mathcal{N}(0, \sigma^2)$ denote the Gaussian distribution of mean 0 and standard deviation σ . We denote its probability density function by $\phi \mapsto p(\phi; \sigma)$, for $\phi \in \mathbb{R}$. Consider the *wrapped* Gaussian distribution $\Psi_{\sigma, q}$ resulting from wrapping the Gaussian distribution around a circle of circumference $q > 0$. Its probability density function $g(\theta; \sigma, q)$ is given by

$$g(\theta; \sigma, q) := \sum_{\ell=-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(\theta + \ell q)^2}{2\sigma^2}\right], \quad \text{for } \theta \in \left[-\frac{q}{2}, \frac{q}{2}\right]. \quad (1)$$

Note that $\Psi_{\sigma, 2\pi}$ is the standard wrapped normal distribution obtained by wrapping $\mathcal{N}(0, \sigma^2)$ around the unit circle, used for instance in directional statistics [39].

LWE schemes use a discretization of a Gaussian over \mathbb{Z}_q . There are two variants of LWE that we will consider in this paper. We will see that we obtain similar results for both distributions. In the initial version by Regev [43], the noise in LWE was a *rounded Gaussian distribution*. This is also what is considered in [1, 29]. Such a distribution can be obtained by sampling from $\Psi_{\sigma, q}$ and rounding the result to the nearest integer in the interval $\left]-\frac{q}{2}, \frac{q}{2}\right]$. We denote this distribution by $\bar{\Psi}_{\sigma, q}$. Its probability mass function is given by

$$\Pr[x \leftarrow \bar{\Psi}_{\sigma, q}] = \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} g(\theta; \sigma, q) d\theta, \quad (2)$$

for x an integer in the interval $\left]-\frac{q}{2}, \frac{q}{2}\right]$.

The LWE problem is believed to be hard when $\sigma \geq \sqrt{k}$ and $q \in \text{poly}(k)$.

The second Gaussian distribution used for LWE is the *discrete Gaussian distribution* $D_{\sigma, q}$. This distribution is used in most of the applications and in the classical LWE reduction [16]. This distribution is, for x an integer in $\left]-\frac{q}{2}, \frac{q}{2}\right]$:

$$\Pr[x \leftarrow D_{\sigma, q}] = \frac{\exp(-x^2/(2\sigma^2))}{\sum_{y \in \left]-\frac{q}{2}, \frac{q}{2}\right]} \exp(-y^2/(2\sigma^2))}. \quad (3)$$

2.4 The LWR Problem

In this section, we define the LWR problem.

Definition 3 (Rounding Function). Let $q \geq p \geq 2$ be positive integers. The LWR problem uses the rounding function from $\mathbb{Z}_q = \{0, \dots, q-1\}$ to $\mathbb{Z}_p = \{0, \dots, p-1\}$, given by²

² For the second component returned by the LWR oracle, we decided to return the rounding of $\langle \mathbf{a}, \mathbf{s} \rangle$ instead of the usual $\lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rfloor$. The problem is equivalent (see, e.g., [3]). However, if we would use the floor operation, the noise in Lemma 19 would not have zero mean but mean $(1/2 - \gcd(p, q)/2q)$ and we would have to introduce tedious correcting terms in (32).

$$\lceil \cdot \rceil_p : \mathbb{Z}_q \rightarrow \mathbb{Z}_p : x \mapsto \left\lceil \left(\frac{p}{q} \right) \cdot x \right\rceil .$$

Definition 4 (LWR Oracle). Let k and $q \geq p \geq 2$ be positive integers. A Learning with Rounding (LWR) oracle $\Lambda_{s,p}$ for a hidden vector $s \in \mathbb{Z}_q^k$, $s \neq 0$ is an oracle returning

$$\left(\mathbf{a} \xleftarrow{U} \mathbb{Z}_q^k, \lceil \langle \mathbf{a}, s \rangle \rceil_p \right) .$$

Definition 5 (Search-LWR). The Search-LWR problem is the problem of recovering the hidden secret s given n queries $(\mathbf{a}^{(j)}, c^{(j)}) \in \mathbb{Z}_q^k \times \mathbb{Z}_p$ obtained from $\Lambda_{s,p}$.

Two reductions from LWE to LWR exist: one with exponential parameters and another with a limited number of samples.

Theorem 6 (Theorem 3.2 in [8]). Let $\beta \in \mathbb{R}_+$ and let χ be any efficiently sampleable distribution over \mathbb{Z} such that $\Pr_{x \leftarrow \chi}[|x| > \beta]$ is negligible. Let $q \geq p \cdot \beta \cdot k^{\omega(1)}$. Then, solving decision-LWR with secrets of size k and parameters p and q is at least as hard as solving decision-LWE over \mathbb{Z}_q with secret of size k and noise distribution χ .

The second result reduces this explosion in the parameters but limits the number of samples the adversary is allowed to get from the LWR oracle.

Theorem 7 (Theorem 4.1 from [3]). Let λ be the security parameter. Let k, ℓ, m, p, γ be positive integers, p_{\max} be the largest prime divisor of q , and $p_{\max} \geq 2\beta\gamma kmp$. Let χ be a probability distribution over \mathbb{Z} such that $\mathbb{E}[|\chi|] \leq \beta$. Then, if $k \geq (\ell + \lambda + 1) \log(q) / \log(2\gamma) + 2\lambda$ and if $\gcd(q, q/p_{\max}) = 1$, the decision-LWR with secret of size k , parameters p and q and limited to m queries is at least as hard as solving decision-LWE over \mathbb{Z}_q with secrets of size ℓ , noise distribution χ and limited to m queries.

2.5 Discrete Fourier Transform

Let p_1, \dots, p_b be integers and let $\theta_{p_j} := \exp(2\pi i / p_j)$, for $1 \leq j \leq b$ and where $i = \sqrt{-1}$. Define the group $G := \mathbb{Z}_{p_1} \times \dots \times \mathbb{Z}_{p_b}$. We may write an element $x \in G$ as (x_1, \dots, x_b) . The discrete Fourier transform (DFT) of a function $f: G \rightarrow \mathbb{C}$ is a function $\hat{f}: G \rightarrow \mathbb{C}$ defined as

$$\hat{f}(\alpha) := \sum_{x \in G} f(x) \theta_{p_1}^{-\alpha_1 x_1} \dots \theta_{p_b}^{-\alpha_b x_b} . \quad (4)$$

The discrete Fourier transform can be computed in time $O(|G| \log(|G|)) =: C_{\text{FFT}} \cdot |G| \log(|G|)$ for a small constant C_{FFT} .

2.6 Hoeffding's Inequality

We will use the following Hoeffding bound.

Theorem 8 ([33]). *Let X_1, X_2, \dots, X_n be n independent random variables such that $\Pr[X_j \in [\alpha_j, \beta_j]] = 1$ for $1 \leq j \leq n$. We define $X = X_1 + \dots + X_n$ and $\mathbb{E}[X]$ to be the expected value of X . We have that*

$$\Pr[X - \mathbb{E}[X] \geq t] \leq \exp\left(\frac{-2t^2}{\sum_{j=1}^n (\beta_j - \alpha_j)^2}\right)$$

and

$$\Pr[X - \mathbb{E}[X] \leq -t] \leq \exp\left(\frac{-2t^2}{\sum_{j=1}^n (\beta_j - \alpha_j)^2}\right),$$

for any $t > 0$.

3 The BKW Algorithm

The BKW algorithm [11], introduced by Blum et al., was the first sub-exponential algorithm given for solving the *Learning Parity with Noise* (LPN) problem. Asymptotically, it has a time and samples complexity of $2^{O(k/\log k)}$. Since LPN can be seen as a special case of LWE where we work over \mathbb{Z}_2 , the BKW algorithm can be adapted to solve Search-LWE over \mathbb{Z}_q with an asymptotic sample and time complexity of $q^{O(k/\log(k))} = 2^{O(k)}$ when the modulus q is polynomial in k [1, 43, 44].

The BKW algorithm can be described as a variant of the standard Gaussian elimination procedure, where a row addition results in the elimination of a whole block of elements instead of a single element. The main idea is that by using ‘few’ row additions and no row multiplications, we limit the size of the noise at the end of the reduction, allowing us to recover a small number of elements of \mathbf{s} with high probability through maximum likelihood. The main complexity drawback of the algorithm comes from finding samples colliding on a block of elements such that their addition eliminates multiple elements at once.

The BKW algorithm takes two integer parameters, usually denoted a and b , such that $a = \lceil k/b \rceil$. The algorithm repeatedly eliminates blocks of up to b elements per row addition, over a rounds, to obtain the samples used for recovering elements of \mathbf{s} . Minimizing the complexity of the algorithm requires a tradeoff between the two parameters. For small a , the reduced samples have low noise and the complexity of recovering elements of \mathbf{s} with high probability is reduced. For large b however, the complexity of finding colliding samples increases.

In [1], Albrecht et al. view the BKW algorithm as a linear system solving algorithm consisting of three stages, denoted *sample reduction*, *hypothesis testing* and *back substitution*. For convenience, we briefly describe each of these stages below.

Sample Reduction. Given an LWE oracle $\Pi_{\mathbf{s}, \chi}$, the goal of this stage is to construct a series of oracles $\mathcal{A}_{\mathbf{s}, \chi, \ell}$, each of which produces samples (\mathbf{a}, c) , where the first $b \cdot \ell$ elements of \mathbf{a} are zero. To create the oracles $\mathcal{A}_{\mathbf{s}, \chi, \ell}$ for $0 < \ell < a$, Albrecht et al. make use of a set of tables T^ℓ , which are maintained throughout the execution of the algorithm. To sample from $\mathcal{A}_{\mathbf{s}, \chi, 1}$, we query the oracle $\mathcal{A}_{\mathbf{s}, \chi, 0}$ (which is the original LWE oracle) to obtain samples (\mathbf{a}, c) to be stored in table T^1 . If T^1 already contains a sample (\mathbf{a}', c') such that \mathbf{a} and $\pm \mathbf{a}'$ agree on their first b coordinates, we do not store (\mathbf{a}, c) but instead output $(\mathbf{a} \mp \mathbf{a}', c \mp c')$. If a sample from $\mathcal{A}_{\mathbf{s}, \chi, 0}$ already has its first b elements to be zero, we directly output it as a sample from $\mathcal{A}_{\mathbf{s}, \chi, 1}$.

For $1 < \ell < a$, we proceed recursively by populating a table T^ℓ of non-zero samples from $\mathcal{A}_{\mathbf{s}, \chi, \ell-1}$ and outputting a query as soon as we get a collision in the table.

Exploiting the symmetry of \mathbb{Z}_q and the fact that we do not need to store queries which are already all-zero on a block, a table T^ℓ contains at most $(q^b - 1)/2$ samples. Then, to create m samples from $\mathcal{A}_{\mathbf{s}, \chi, \ell}$, we will need at most $m + \frac{q^b - 1}{2}$ calls to $\mathcal{A}_{\mathbf{s}, \chi, \ell-1}$. Furthermore, since there is no use in storing the zero elements from reduced samples, table T^ℓ stores samples of size $n - (\ell - 1) \cdot b + 1$ elements from \mathbb{Z}_q . The description of the oracles $\mathcal{A}_{\mathbf{s}, \chi, \ell}$ is given in Algorithm 1.

In the original BKW algorithm (see [11, 35]), one would then take samples from $\mathcal{A}_{\mathbf{s}, \chi, a-1}$, i.e., samples with zeros everywhere except in the first b positions, and delete any sample (\mathbf{a}, c) with more than one non-zero coordinate \mathbf{a}_i . The remaining samples would be used to recover one bit of \mathbf{s} at a time.

Albrecht et al. generalized a bit the result. Instead of keeping only one single element of the secret vector, they select a parameter $d \leq k - (a - 1) \cdot b$ and create a final oracle $\mathcal{A}_{\mathbf{s}, \chi, a}$, which produces samples with d non-zero entries at fixed positions in \mathbf{a} . These samples are used to recover d bits of s through exhaustive search over q^d values. The oracle $\mathcal{A}_{\mathbf{s}, \chi, a}$ is defined similarly as above, making use of a final table T^a . It samples from $\mathcal{A}_{\mathbf{s}, \chi, a-1}$ and adds (or subtracts) queries (\mathbf{a}, c) , (\mathbf{a}', c') , for which \mathbf{a} and $\pm \mathbf{a}'$ agree on coordinates $(a - 1) \cdot b + 1$ through $k - d - 1$. Albrecht et al. note that they obtain the best results when choosing d equal to 1 or 2. Note that $d = 1$ corresponds to the BKW algorithm.³

Hypothesis Testing. After the reduction phase, Albrecht et al. are left with samples (\mathbf{a}, c) from $\mathcal{A}_{\mathbf{s}, \chi, a}$, where \mathbf{a} has d non-zero elements. We can view $\mathcal{A}_{\mathbf{s}, \chi, a}$ as outputting samples in $\mathbb{Z}_q^d \times \mathbb{Z}_q$. Let \mathbf{s}' denote the d first elements of \mathbf{s} . Since \mathbf{a} was obtained by summing or subtracting up to 2^a samples from the LWE oracle $\Pi_{\mathbf{s}, \chi}$ (and considering the fact that χ is symmetric around 0), the noise $(c - \langle \mathbf{a}, \mathbf{s}' \rangle)$ of the reduced samples follows the distribution of the sum of 2^a noise samples. The problem of recovering \mathbf{s}' can then be seen as a problem of distinguishing between the noise distributions for \mathbf{s}' and $\mathbf{v} \neq \mathbf{s}'$.

³ The only difference between the two algorithms is that the original BKW algorithm restarts every time $\mathcal{A}_{\mathbf{s}, \chi, a}$ outputs something.

Algorithm 1. Oracle $\mathcal{A}_{\mathbf{s}, \chi, \ell}$, for $0 < \ell < a$

State: A table T^ℓ (initially empty)
Output: An LWE tuple (\mathbf{a}, c) such that \mathbf{a} has the first $b \cdot \ell$ elements set to 0.

```

1: loop
2:   Let  $(\mathbf{a}, c) \leftarrow \mathcal{A}_{\mathbf{s}, \chi, \ell-1}$ .
3:   if  $\mathbf{a}$  has the first  $b \cdot \ell$  elements set to 0 then
4:     return  $(\mathbf{a}, c)$ .
5:   end if
6:   if there is  $(\mathbf{a}', c') \in T^\ell$  such that  $\mathbf{a}$  and  $\pm \mathbf{a}'$  are equal on the first  $b \cdot \ell$  positions
    then
7:     return  $(\mathbf{a} \mp \mathbf{a}', c \mp c')$ 
8:   end if
9:   Add  $(\mathbf{a}, c)$  to  $T^\ell$ .
10: end loop
```

By performing an exhaustive search over \mathbb{Z}_q^d and making use of the log-likelihood ratio, Albrecht et al. determine the number m of samples from $\mathcal{A}_{\mathbf{s}, \chi, a}$ which should be required to recover \mathbf{s}' with high enough probability.

As already mentioned, the analysis of the solving phase from [1] makes use of the heuristic assumption that the noise contributions of the samples from $\mathcal{A}_{\mathbf{s}, \chi, a}$ are independent and that the sum of rounded Gaussians also follows a rounded Gaussian distribution.

Back Substitution. This stage was not part of the original BKW algorithm for LPN [11, 35] (which does not make use of the set of tables T defined previously either). It is analogous to the back substitution typically used in Gaussian elimination and is a clever way of reducing the size of the LWE problem after part of the secret \mathbf{s} has been recovered.

Indeed, once d elements of \mathbf{s} are recovered with high probability, we can perform a back substitution over the set of tables T , zeroing-out d elements in each sample. To recover the next d elements from \mathbf{s} , we query m new samples from $\Pi_{\mathbf{s}, \chi}$ and reduce them through the tables T (which are already filled) to obtain samples for hypothesis testing. Note that as soon as we recover all the bits at positions $(\ell-1) \cdot b$ through $\ell \cdot b - 1$, the oracle $\mathcal{A}_{\mathbf{s}, \chi, \ell}$ and its corresponding table T^ℓ become superfluous and further samples will need one reduction phase less.

4 The LWE-solving Algorithm

In this section, we present our new LWE-solving algorithm. Following the structure from [1], our algorithm will also consist of the sample reduction, hypothesis testing and back substitution phases. However, we change the hypothesis testing phase with an idea similar to the LF1 algorithm [35]. Indeed, since the Walsh-Hadamard transform can be seen as a multidimensional discrete Fourier transform in \mathbb{Z}_2 , it would seem plausible that a similar optimization could be achieved

over \mathbb{Z}_q for LWE. As we have seen, the BKW algorithm for LWE from [1] differs slightly from the original BKW algorithm in its reduction phase. Recall that after reducing samples to a block of size $k' \leq b$, Albrecht et al. further reduce the samples to d elements. Our idea is to remove this last reduction to d elements and recover directly the k' elements of \mathbf{s} using a DFT. Thus, the samples we use for the DFT would have noise sampled from the sum of 2^{a-1} discretized Gaussians instead of 2^a , which might also lead to a significant improvement. As for most other works on LPN or LWE solving algorithms, we will make use of an heuristic assumption of independence for the noise of the reduced samples.

Finally, note that the LF1 algorithm uses the exact same reduction phase as the original BKW. Similarly, our algorithm will use (nearly) the same reduction phase as in [1], combined with a different hypothesis testing phase. The major differences in our reduction phase will be that we perform one reduction round less, and that we decide to store and re-use samples for solving successive blocks of \mathbf{s} .

4.1 Sample Reduction

As mentioned previously, our algorithm uses the same reduction phase as the BKW algorithm from [1], except that we always stop the reduction as soon as we reach a block of $k' \leq b$ non-zero elements. We will construct the oracles $\mathcal{A}_{\mathbf{s}, \chi, \ell}$ and the tables T^ℓ only for $1 \leq \ell \leq a-1$. It is thus fairly trivial to adapt the results from [1] to bound the complexity of our algorithm's reduction phase.

Lemma 9 (Lemma 2 and 3 from [1]). *Let k, q be positive integers and $\Pi_{\mathbf{s}, \chi}$ be an LWE oracle, where $\mathbf{s} \in \mathbb{Z}_q^k$. Let $a \in \mathbb{Z}$ with $1 \leq a \leq k$, let b be such that $ab \leq k$, and let $k' = k - (a-1)b$. The worst case cost of obtaining m samples (\mathbf{a}_i, c_i) from the oracle $\mathcal{A}_{\mathbf{s}, \chi, a-1}$, where the \mathbf{a}_i are zero for all but the first k' elements, is upper bounded by*

$$\left(\frac{q^b - 1}{2}\right) \left(\frac{(a-1) \cdot (a-2)}{2} (k+1) - \frac{ab \cdot (a-1) \cdot (a-2)}{6} \right) + m \left(\frac{a-1}{2} (k+2) \right)$$

additions in \mathbb{Z}_q and $(a-1) \cdot \frac{q^b - 1}{2} + m$ calls to $\Pi_{\mathbf{s}, \chi}$.

The memory required in the worst case to store the set of tables T^1 through T^{a-1} , expressed in elements of \mathbb{Z}_q is upper bounded by

$$\left(\frac{q^b - 1}{2} \cdot (a-1) \cdot \left(k + 1 - b \frac{a-2}{2} \right) \right).$$

Proof. The proof follows exactly the one from [1], with the exception that we do not use any table T^a . \square

4.2 Hypothesis Testing

At the end of the reduction phase, we are left with m samples $(\mathbf{a}^{(j)}, c^{(j)})$ from the oracle $\mathcal{A}_{s,\chi,a-1}$, where each $\mathbf{a}^{(j)}$ has all elements equal to zero except for a block of size $k' = k - (a-1) \cdot b$. Let \mathbf{s}' denote the corresponding block of the secret \mathbf{s} . We can view the oracle $\mathcal{A}_{s,\chi,a-1}$ as returning samples in $\mathbb{Z}_q^{k'} \times \mathbb{Z}_q$. We will consider that each such sample is the sum of 2^{a-1} samples (or their negation) from the LWE oracle $\Pi_{s,\chi}$. Then, the noise $\langle \mathbf{a}^{(j)}, \mathbf{s}' \rangle - c^{(j)}$ will correspond to the sum of 2^{a-1} independent samples from the distribution χ , multiplied by ± 1 , and taken modulo q . We perform our analysis when χ is the discrete Gaussian distribution (3) and when χ is the rounded Gaussian distribution (2) which are used in most of the LWE research, i.e., we let $\chi = D_{\sigma,q}$ or $\chi = \bar{\Psi}_{\sigma,q}$.

We represent our m samples as a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times k'}$ with rows \mathbf{A}_j and a vector $\mathbf{c} \in \mathbb{Z}_q^m$. Recall that $\theta_q := \exp(2\pi i/q)$. Let us consider the function

$$f(\mathbf{x}) := \sum_{j=1}^m \mathbb{1}_{\{\mathbf{A}_j=\mathbf{x}\}} \theta_q^{c_j}, \quad \forall \mathbf{x} \in \mathbb{Z}_q^{k'}. \quad (5)$$

The discrete Fourier transform of f is

$$\widehat{f}(\boldsymbol{\alpha}) := \sum_{\mathbf{x} \in \mathbb{Z}_q^{k'}} f(\mathbf{x}) \theta_q^{-\langle \mathbf{x}, \boldsymbol{\alpha} \rangle} = \sum_{\mathbf{x} \in \mathbb{Z}_q^{k'}} \sum_{j=1}^m \mathbb{1}_{\{\mathbf{A}_j=\mathbf{x}\}} \theta_q^{c_j} \theta_q^{-\langle \mathbf{x}, \boldsymbol{\alpha} \rangle} = \sum_{j=1}^m \theta_q^{-((\langle \mathbf{A}_j, \boldsymbol{\alpha} \rangle - c_j))}.$$

In particular, note that

$$\widehat{f}(\mathbf{s}') = \sum_{j=1}^m \theta_q^{-((\langle \mathbf{A}_j, \mathbf{s}' \rangle - c_j))} = \sum_{j=1}^m \theta_q^{-(\nu_{j,1} \pm \dots \pm \nu_{j,2^a-1})}, \quad (6)$$

where the $\nu_{j,l}$ are independent samples from χ . Note that we dropped the reduction of the sum of the ν modulo q , since $\theta_q^{kq} = 1$, for $k \in \mathbb{Z}$.

We will now show, through a series of lemmas, that for appropriate values for m and a , the maximum value of the function $\text{Re}(\widehat{f}(\boldsymbol{\alpha}))$ is reached by \mathbf{s}' with high probability. Our algorithm for recovering \mathbf{s}' will thus consist in finding the highest peak of the real part of the DFT of $f(\mathbf{x})$.

We start first with two technical lemmas regarding Gaussian distributions which might be of independent interest.

Lemma 10. *For q an odd integer, let $X \sim \bar{\Psi}_{\sigma,q}$ and let $Y \sim 2\pi X/q$. Then*

$$\mathbb{E}[\cos(Y)] \geq \frac{q}{\pi} \sin\left(\frac{\pi}{q}\right) e^{-2\pi^2 \sigma^2 / q^2} \quad \text{and} \quad \mathbb{E}[\sin(Y)] = 0.$$

Proof. Let S_ℓ be the set of integers in $] -q/2 + \ell q, q/2 + \ell q]$. Using (1) and (2), we can write

$$\mathbb{E}[\cos(Y)] = \sum_{x \in S_0} \cos\left(\frac{2\pi}{q}x\right) \sum_{\ell=-\infty}^{\infty} \int_{x-1/2}^{x+1/2} p(\theta + \ell q; \sigma) d\theta \quad (7)$$

$$= \sum_{\ell=-\infty}^{\infty} \sum_{x \in S_0} \cos\left(\frac{2\pi}{q}x + 2\pi\ell\right) \int_{x-1/2}^{x+1/2} p(\theta + \ell q; \sigma) d\theta \quad (8)$$

$$= \sum_{\ell=-\infty}^{\infty} \sum_{x \in S_0} \cos\left(\frac{2\pi}{q}(x + \ell q)\right) \int_{x-1/2+\ell q}^{x+1/2+\ell q} p(\theta; \sigma) d\theta \quad (9)$$

$$= \sum_{\ell=-\infty}^{\infty} \sum_{x' \in S_\ell} \cos\left(\frac{2\pi}{q}x'\right) \int_{x'-1/2}^{x'+1/2} p(\theta; \sigma) d\theta \quad (10)$$

$$= \sum_{x'=-\infty}^{\infty} \cos\left(\frac{2\pi}{q}x'\right) \int_{x'-1/2}^{x'+1/2} p(\theta; \sigma) d\theta \quad (11)$$

$$= \sum_{\chi=-\infty}^{\infty} \mathcal{F}\left(\cos\left(x\frac{2\pi}{q}\right) \int_{x-1/2}^{x+1/2} p(\theta; \sigma) d\theta\right)(\chi), \quad (12)$$

where, for (10), we used $x' := x + \ell q$ and, for (12), we used the Poisson summation formula (Lemma 25 in Appendix A). Basics about continuous Fourier transforms can be found in Appendix A. The Fourier transforms of $\cos(2\pi x/q)$ and $1/(\sigma\sqrt{2\pi}) \exp[-x/(2\sigma^2)]$ can be found in Appendix A. We are now ready to prove the lemma (we drop some (χ) for readability). For integer values of χ , we have

$$\mathcal{F}\left(\cos\left(x\frac{2\pi}{q}\right) \int_{x-1/2}^{x+1/2} \frac{1}{\sigma\sqrt{2\pi}} e^{-\theta^2/(2\sigma^2)} d\theta\right) \quad (13)$$

$$= \mathcal{F}\left(\cos\left(x\frac{2\pi}{q}\right)\right) * \left(\mathcal{F}\left(\int_{-\infty}^{x+\frac{1}{2}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\theta^2}{2\sigma^2}} d\theta\right) - \mathcal{F}\left(\int_{-\infty}^{x-\frac{1}{2}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\theta^2}{2\sigma^2}} d\theta\right)\right) \quad (14)$$

$$= \mathcal{F}\left(\cos\left(x\frac{2\pi}{q}\right)\right) * \left(\left(e^{\pi i \chi} - e^{-\pi i \chi}\right) \mathcal{F}\left(\int_{-\infty}^x \frac{1}{\sigma\sqrt{2\pi}} e^{-\theta^2/(2\sigma^2)} d\theta\right)\right) \quad (15)$$

$$= \frac{1}{2} \left(\delta\left(\chi - \frac{1}{q}\right) + \delta\left(\chi + \frac{1}{q}\right)\right) * \left(\left(e^{\pi i \chi} - e^{-\pi i \chi}\right) \left(\frac{1}{2\pi i \chi} e^{-2\pi^2 \sigma^2 x^2} + \frac{1}{2} \delta(\chi)\right)\right) \quad (16)$$

$$= \frac{1}{2} \left(\delta\left(\chi - \frac{1}{q}\right) + \delta\left(\chi + \frac{1}{q}\right)\right) * \left(\sin(\pi \chi) \left(\frac{1}{\pi \chi} e^{-2\pi^2 \sigma^2 x^2}\right)\right) \quad (17)$$

$$= \frac{q}{2\pi} \sin\left(\frac{\pi}{q}\right) (-1)^\chi \left(\frac{e^{-2\pi^2 \sigma^2 (qx+1)^2/q^2}}{qx+1} - \frac{e^{-2\pi^2 \sigma^2 (qx-1)^2/q^2}}{qx-1}\right), \quad (18)$$

where (14) is the convolution property of the FT, (15) comes from the translation property of the FT, (16) comes from the integration property of the FT, and (17) holds since $\delta(\chi \pm 1/q) * \delta(\chi) = 0$ for integer values of χ . We can write (12) as

$$\begin{aligned} & \frac{q}{\pi} \sin\left(\frac{\pi}{q}\right) \exp^{-2\pi^2\sigma^2/q^2} \\ & + \sum_{\chi=1}^{\infty} \frac{q}{\pi} \sin\left(\frac{\pi}{q}\right) (-1)^{\chi} \left(\frac{e^{-2\pi^2\sigma^2(q\chi+1)^2/q^2}}{q\chi+1} - \frac{e^{-2\pi^2\sigma^2(q\chi-1)^2/q^2}}{q\chi-1} \right). \end{aligned}$$

Notice that the sum term in this equation is alternating and decreasing in absolute value when χ grows (derivative is negative). Notice also that the first term (when $\chi = 1$) is positive. Hence this sum is greater than 0 and we get our result for $\mathbb{E}[\cos(Y)]$.

For $\mathbb{E}[\sin(Y)]$, note that when q is odd, X and Y are perfectly symmetric around 0. The result then follows trivially from the symmetry of the sine function. \square

Lemma 11. *For q an odd integer, let $X \sim D_{\sigma,q}$ and let $Y \sim 2\pi X/q$. Then*

$$\mathbb{E}[\cos(Y)] \geq 1 - \frac{2\pi^2\sigma^2}{q^2} \quad \text{and} \quad \mathbb{E}[\sin(Y)] = 0.$$

Proof. Using [7, Lemma 1.3] with $a = 1/(2\sigma^2)$, we have that $\mathbb{E}[X^2] \leq \sigma^2$. Hence, using $\cos(x) \geq 1 - x^2/2$,

$$\mathbb{E}[\cos(2\pi X/q)] \geq 1 - 2\pi^2\mathbb{E}[X^2]/q^2 = 1 - 2\pi^2\sigma^2/q^2.$$

For $\mathbb{E}[\sin(Y)]$, note that when q is odd, X and Y are perfectly symmetric around 0. The result then follows trivially from the symmetry of the sine function. \square

Definition 12 ($R_{\sigma,q,\chi}$). *In the following, let $R_{\sigma,q,\chi} := \mathbb{E}[\cos(\chi)]$, i.e.,*

$$R_{\sigma,q,\chi} := \begin{cases} \frac{q}{\pi} \sin\left(\frac{\pi}{q}\right) e^{-2\pi^2\sigma^2/q^2} & \text{when } \chi = \bar{\Psi}_{q,\sigma} \\ 1 - \frac{2\pi^2\sigma^2}{q^2} & \text{when } \chi = D_{q,\sigma} \end{cases}$$

Lemma 13. $\mathbb{E}[\operatorname{Re}(\widehat{f}(\mathbf{s}'))] \geq m \cdot (R_{\sigma,q,\chi})^{2^{a-1}}$.

Proof. From (6), we get

$$\mathbb{E}[\operatorname{Re}(\widehat{f}(\mathbf{s}'))] = \operatorname{Re} \left(\sum_{j=1}^m \mathbb{E} \left[\theta_q^{-(\nu_{j,1} \pm \dots \pm \nu_{j,2^{a-1}})} \right] \right) = \operatorname{Re} \left(\sum_{j=1}^m \mathbb{E} \left[\cos \left(\frac{2\pi}{q} \nu_{j,1} \right) \right]^{2^{a-1}} \right),$$

using the independence of the noise samples $\nu_{j,\ell}$ and $\mathbb{E}[\theta_q^{\pm\nu_{j,\ell}}] = \mathbb{E}[\cos(2\pi\nu_{j,\ell}/q)]$ (which follows from Lemmas 10 and 11). Using Lemmas 10 and 11 again, we have that $\mathbb{E}[\cos(2\pi\nu_{j,\ell}/q)] \geq R_{\sigma,q,\chi}$. Hence, we get that

$$\mathbb{E}[\operatorname{Re}(\widehat{f}(\mathbf{s}'))] > \sum_{j=1}^m (R_{\sigma,q,\chi})^{2^{a-1}} = m \cdot (R_{\sigma,q,\chi})^{2^{a-1}}. \quad (19)$$

\square

Lemma 14. Let $G \subseteq \mathbb{Z}_q$ be a subgroup of \mathbb{Z}_q , let $X \xleftarrow{U} G$ and let $e \in \mathbb{Z}_q$ be independent from X . Then, $\mathbb{E}[\theta_q^{X+e}] = 0$.

Proof. Define $Y = \frac{2\pi}{q}X$. Then Y is a random variable following a discrete uniform distribution on the unit circle. Then $\mathbb{E}[\theta_q^X] = 0$ follows from the analysis of discrete circular uniform distributions (see e.g. [6]). Now, since X and e are independent, $\mathbb{E}[\theta_q^{X+e}] = \mathbb{E}[\theta_q^X]\mathbb{E}[\theta_q^e] = 0$. \square

Lemma 15. $\arg \max_{\alpha} \operatorname{Re}(\hat{f}(\alpha)) = s'$ with probability greater than

$$1 - q^{k'} \cdot \exp\left(-\frac{m}{8} \cdot (R_{\sigma,q,\chi})^{2^a}\right).$$

Proof. A similar proof is proposed for LPN in [12]. We are looking to upper bound the probability that there is some $\alpha \neq s'$ such that $\operatorname{Re}(\hat{f}(\alpha)) \geq \operatorname{Re}(\hat{f}(s'))$. Using a union bound, we may upper bound this by $q^{k'}$ times the probability that $\operatorname{Re}(\hat{f}(\alpha)) \geq \operatorname{Re}(\hat{f}(s'))$ for some fixed vector $\alpha \in \mathbb{Z}_q^{k'}$, $\alpha \neq s'$ which is the probability that

$$\sum_{j=1}^m \left(\operatorname{Re}(\theta_q^{-((A_j, s') - c_j)}) - \operatorname{Re}(\theta_q^{-((A_j, \alpha) - c_j)}) \right) \leq 0.$$

Let $y = \alpha - s' \in \mathbb{Z}_q^{k'}$. Also, define $e_j := \langle A_j, s' \rangle - c_j$, for $1 \leq j \leq m$. Then, $\langle A_j, \alpha \rangle - c_j = \langle A_j, y \rangle + e_j$. Note that since A_j is uniformly distributed at random, independently from e_j , and y is fixed and non-zero, $\langle A_j, y \rangle$ is uniformly distributed in a subgroup of \mathbb{Z}_q , and thus so is $\langle A_j, \alpha \rangle - c_j$. Hence, we can apply Lemma 14.

From our heuristic assumption, we will consider X_1, X_2, \dots, X_m to be independent random variables with $X_j = u_j - v_j$, where

$$u_j = \operatorname{Re}(\theta_q^{-((A_j, s') - c_j)}) \quad \text{and} \quad v_j = \operatorname{Re}(\theta_q^{-((A_j, \alpha) - c_j)}). \quad (20)$$

Note that $X_j \in [-2, 2]$ for all j . Furthermore, let $X = \sum_{j=1}^m X_j$. Using Lemmas 13 (for the u_j 's) and 14 (for the v_j 's), we get that

$$\mathbb{E}[X] \geq m \cdot (R_{\sigma,q,\chi})^{2^{a-1}}. \quad (21)$$

We will bound the probability that $X \leq 0$ using Hoeffding's inequality (Theorem 8). Let $t = \mathbb{E}[X] > 0$. Then,

$$\begin{aligned} \Pr[X \leq 0] &= \Pr[(X - \mathbb{E}[X]) \leq -\mathbb{E}[X]] \leq \exp\left(\frac{-2(\mathbb{E}[X])^2}{16m}\right) \\ &\leq \exp\left(-\frac{m}{8} \cdot (R_{\sigma,q,\chi})^{2^a}\right). \end{aligned} \quad (22)$$

Applying the aforementioned union-bound, we get the desired result. \square

We are now ready to derive the number of samples m required to recover the correct secret block \mathbf{s}' with high probability.

Theorem 16. *Let k, q be positive integers and $\Pi_{\mathbf{s}, \chi}$ be an LWE oracle, where $\mathbf{s} \in \mathbb{Z}_q^k$. Let $a \in \mathbb{Z}$ with $1 \leq a \leq k$, let b be such that $ab \leq k$, and let $k' = k - (a-1)b$. Let $\mathcal{A}_{\mathbf{s}, \chi, a-1}$ be the oracle returning samples (\mathbf{a}_i, c_i) where the \mathbf{a}_i are zero for all but the first k' elements. Denote the vector consisting of the first k' elements of \mathbf{s} as \mathbf{s}' . Fix an $\epsilon \in (0, 1)$. Then, the number of independent samples m^{LWE} from $\mathcal{A}_{\mathbf{s}, \chi, a-1}$, which are required such that we fail to recover the secret block \mathbf{s}' with probability at most ϵ satisfies*

$$m^{LWE} \geq \begin{cases} 8 \cdot k' \cdot \log\left(\frac{q}{\epsilon}\right) \cdot \left(\frac{q}{\pi} \sin\left(\frac{\pi}{q}\right) e^{-2\pi^2\sigma^2/q^2}\right)^{-2^a} & \text{when } \chi = \bar{\Psi}_{\sigma, q} \\ 8 \cdot k' \cdot \log\left(\frac{q}{\epsilon}\right) \cdot \left(1 - \frac{2\pi^2\sigma^2}{q^2}\right)^{-2^a} & \text{when } \chi = D_{\sigma, q}. \end{cases}$$

Furthermore, the hypothesis testing phase (the FFT phase in Algorithm 2) that recovers \mathbf{s}' requires $2m^{LWE} + C_{\text{FFT}} \cdot k' \cdot q^{k'} \cdot \log q$ operations in \mathbb{C} and requires storage for $q^{k'}$ complex numbers, where C_{FFT} is the small constant in the complexity of the FFT.⁴

Proof. For a fixed m , we get

$$\epsilon = \Pr \left[\exists \boldsymbol{\alpha} \neq \mathbf{s}' : \operatorname{Re}(\hat{f}(\boldsymbol{\alpha})) \geq \operatorname{Re}(\hat{f}(\mathbf{s}')) \right] < q^{k'} \cdot \exp\left(-\frac{m}{8} \cdot (R_{\sigma, q, \chi})^{2^a}\right).$$

Solving for m , we get the desired result.

Concerning the algorithmic and memory complexities, we need to store the values of the function $f(\mathbf{x})$ as $q^{k'}$ elements from \mathbb{C} . For each of the m^{LWE} samples we receive from $\mathcal{A}_{\mathbf{s}, \chi, a-1}$, we compute an exponentiation and an addition in \mathbb{C} to update $f(\mathbf{x})$ and then discard the sample. Finally, computing the discrete Fourier transform of f can be achieved with $C_{\text{FFT}} \cdot k' \cdot q^{k'} \cdot \log q$ complex operations, and no additional memory, using an in-place FFT algorithm. \square

The hypothesis testing part of the algorithm is summarized in Algorithm 2.

4.3 Back Substitution

We use a similar back substitution mechanism as the one described in [1]. Note that we have to apply back substitution on one table less, since we performed only $a - 1$ reductions. Furthermore, since we recovered a complete block of \mathbf{s} , the table T^{a-1} would be completely zeroed-out by back substitution and can

⁴ One might comment on the required precision needed to compute the DFT. For this, we set our precision to $O\left(\log(m(R_{\sigma, q, \chi})^{2^a})\right)$ bits which is the expected size of our highest peak in the DFT. Using this result along with some standard results about the exact complexity to compute a DFT with a given precision (see, e.g., [18]), the ratio between our (binary) complexities and the binary complexities of [1] remain the same.

Algorithm 2. Hypothesis testing algorithm for LWE.

Input: m independent LWE samples with only $k' := k - (a-1)b$ non-zero components in \mathbf{a} . We represent our samples as a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times k'}$ and a vector $\mathbf{c} \in \mathbb{Z}_q^m$.

Output: A vector consisting of the k' elements of \mathbf{s} that are at the non-zero positions of \mathbf{a}

- 1: Compute the fast Fourier Transform $\widehat{f}(\boldsymbol{\alpha})$ of the function $f(\mathbf{x})$: $\sum_{j=1}^m \mathbf{1}_{\mathbf{A}_j=\mathbf{x}} \theta_q^{c_j}$
- 2: **return** $\arg \max_{\boldsymbol{\alpha} \in \mathbb{Z}_q^{k'}} \widehat{f}(\boldsymbol{\alpha})$

therefore simply be dropped after the hypothesis testing phase. Finally, we do not discard the m^{LWE} queries from $\Pi_{\mathbf{s}, \chi}$, which were reduced and then used for the solving phase. Instead, we store these m^{LWE} original queries and re-use $m' < m^{\text{LWE}}$ of these queries for the next block of \mathbf{s} .

4.4 Complexity of BKW with Multidimensional DFT

We now have all the results we need in order to state the total complexity of solving SEARCH-LWE with our algorithm. For ease of notation, we will consider from here on that the parameters a and b are chosen such that $k = a \cdot b$. Note that the general case, where $k = (a-1) \cdot b + k'$, follows similarly from our previous results.

Theorem 17 (Complexity of SEARCH-LWE). *Let k, q be positive integers and $\Pi_{\mathbf{s}, \chi}$ be an LWE oracle, where $\mathbf{s} \in \mathbb{Z}_q^k$. Let $a, b \in \mathbb{N}$ be such that $a \cdot b = k$. Let C_{FFT} be the small constant in the complexity of the fast Fourier transform computation. Let $0 < \epsilon < 1$ be a targeted success rate and define $\epsilon' := (1 - \epsilon)/a$. For $0 \leq j \leq a-1$, let*

$$m_{j, \epsilon}^{\text{LWE}} := \begin{cases} 8 \cdot b \cdot \log\left(\frac{q}{\epsilon}\right) \cdot \left(\frac{q}{\pi} \sin\left(\frac{\pi}{q}\right) e^{-2\pi^2\sigma^2/q^2}\right)^{-2^{a-j}} & \text{when } \chi = \bar{\Psi}_{\sigma, q} \\ 8 \cdot b \cdot \log\left(\frac{q}{\epsilon}\right) \cdot \left(1 - \frac{2\pi^2\sigma^2}{q^2}\right)^{-2^{a-j}} & \text{when } \chi = D_{\sigma, q} . \end{cases}$$

Under the standard heuristic that all the samples after reduction are independent (which was also used in the previous work), the time complexity of our algorithm to recover the secret \mathbf{s} with probability at least ϵ is $c_1 + c_2 + c_3 + c_4$, where

$$c_1 := \left(\frac{q^b - 1}{2}\right) \cdot \left(\frac{(a-1) \cdot (a-2)}{2}(k+1) - \frac{b}{6}(a \cdot (a-1) \cdot (a-2))\right) \quad (23)$$

is the number of additions in \mathbb{Z}_q to produce all tables T^j , $0 \leq j \leq a-1$,

$$c_2 := \sum_{j=0}^{a-1} m_{j, \epsilon'}^{\text{LWE}} \cdot \frac{a-1-j}{2} \cdot (k+2) \quad (24)$$

is the number of additions in \mathbb{Z}_q to produce the samples required to recover all blocks of \mathbf{s} with probability ϵ ,

$$c_3 := 2 \left(\sum_{j=0}^{a-1} m_{j,\epsilon'}^{\text{LWE}} \right) + C_{\text{FFT}} \cdot k \cdot q^b \cdot \log(q) \quad (25)$$

is the number of operations in \mathbb{C} to prepare and compute the DFTs, and

$$c_4 := (a-1) \cdot (a-2) \cdot b \cdot \frac{q^b - 1}{2} \quad (26)$$

is the number of operations in \mathbb{Z}_q for back substitution.

The number of calls to the oracle $\Pi_{\mathbf{s},\chi}$ is

$$(a-1) \cdot \frac{q^b - 1}{2} + m_{0,\epsilon}^{\text{LWE}}. \quad (27)$$

Finally, the memory complexity in number of elements from \mathbb{Z}_q and \mathbb{C} are respectively

$$\left(\frac{q^b - 1}{2} \cdot (a-1) \cdot \left(k + 1 - b \frac{a-2}{2} \right) \right) + m_{0,\epsilon}^{\text{LWE}} \quad \text{and} \quad q^b. \quad (28)$$

Proof. To recover \mathbf{s} , we need to recover each block of \mathbf{s} successfully. Since we are making use of the same set of tables T and reduced queries for each block, these events are not independent. Using a union bound, and a failure probability bounded by $(1-\epsilon)/a$ for each of the a blocks thus leads to a overall success probability of at least ϵ .

- The cost of constructing the set of tables T in (23) is given by Lemma 9. Note that these tables are constructed only once and maintained throughout the execution of the algorithm.
- As per Lemma 9, the cost of obtaining m samples from the oracle $\mathcal{A}_{\mathbf{s},\chi,a-1}$ is upper bounded by $m \cdot \frac{a-1}{2} \cdot (k+2)$. Noting that after solving the j th block, the table T^j is dropped, the result in (24) follows.
- The DFT has to be applied a times, for each block of size b . Since the number of samples required is updated for each block, we get equation (25).
- After solving the first block, back substitution has to be applied to $a-2$ tables (table T^{a-1} can be dropped). Per table, the substitution has cost $2b$ for each of the $\frac{q^b-1}{2}$ rows. In total, we get a cost of $\sum_{j=1}^{a-2} 2 \cdot b \cdot \left(i \cdot \frac{q^b-1}{2} \right)$, as in (26).
- The required number of oracle samples follows from Lemma 9. Note that the samples needed to fill up the tables are required only once and that the $m_{0,\epsilon}^{\text{LWE}}$ additional queries are stored and can be reused for each block of \mathbf{s} since $m_{0,\epsilon}^{\text{LWE}} > m_{j,\epsilon}^{\text{LWE}}$ for $j > 0$. This gives us the total from (27).

- Finally, the storage cost for the tables follows from Lemma 9. In addition, we need an array of size q^b to store the complex function on which we apply the DFT (we assume an in-place DFT algorithm requiring no extra storage). We also store the $m_{0,\epsilon}^{\text{LWE}}$ samples queried to solve the first block. Combining these results gives us (28).

□

4.5 Using Fewer Samples

If the number of queries to the LWE oracle is limited we can use an idea introduced by Lyubashevsky [37]. The idea is to use a universal family of hash function to combine samples and create new ones. However, these new samples will have higher noise.

Theorem 18. *Let $\epsilon \geq (\log q+1)/\log k$. Then, one can convert an LWE instance $\Pi_{s,\chi}$ where χ is $\bar{\Psi}_{\sigma,q}$ (resp. $D_{\sigma,q}$) and using $k^{1+\epsilon}$ samples into an LWE instance $\Pi_{s,\chi'}$ where χ' is $\bar{\Psi}_{\sigma \lceil (\log q+1)k/(\epsilon \log k) \rceil, q}$ (resp. $D_{\sigma \lceil (\log q+1)k/(\epsilon \log k) \rceil}$) without any sample limit.*

Proof (sketch). The proof is exactly the same as in [37] except for few differences that we state here. We let our samples be $A = \mathbf{a}^{(1)}, \dots, \mathbf{a}^{(k^{1+\epsilon})} \in \mathbb{Z}_q^k$. Let also $X \subset \{0,1\}^{k^{1+\epsilon}}$ with $x \in X$ if $\sum_j x_j = \lceil (\log(q)+1)k/(\epsilon \log k) \rceil$. We use the following universal family of hash function $H := \{h_A : X \rightarrow \mathbb{Z}_q^k\}$ where A is defined above and $h_A(x) := x_1 a^{(1)} + \dots + x_{k^{1+\epsilon}} a^{(k^{1+\epsilon})}$. By the Leftover Hash Lemma [34], when A and x are uniformly distributed, with probability greater than $1 - 2^{-k/4}$, $\Delta(h_A(x), U) \leq 2^{-n/4}$, where U is the uniform probability distribution over \mathbb{Z}_q^k . Note that the Leftover Hash Lemma holds since

$$|X| \geq \left(\frac{k^{1+\epsilon}}{\lceil (\log q+1)k/(\epsilon \log k) \rceil} \right)^{\lceil (\log q+1)k/(\epsilon \log k) \rceil} \geq q^k,$$

when $\epsilon \geq (\log q+1)/\log k$.

□

The LF2 Heuristic. In [35], Levieil and Fouque propose LF2, an heuristic improvement for the reduction phase of their LPN solving algorithm LF1. The main idea of LF2 is to compute the sum (or difference) of *any* pair of samples (\mathbf{a}, c) and (\mathbf{a}', c') , which agree on b particular coordinates. Thus, in an entry of a reduction table T^i , we would store not only one, but all samples agreeing (up to negation) on b coordinates. Then, when reducing a sample (\mathbf{a}, c) , we could output $(\mathbf{a} \pm \mathbf{a}', c \pm c')$ for *each* sample (\mathbf{a}', c') in the corresponding table entry. Note that if we have x samples agreeing on b positions, we can output $\binom{x}{2}$ reduced samples.

An interesting case arises when we take exactly $3 \cdot q^b/2$ oracle samples. In the worst case, we get exactly 3 samples per entry in table T^1 . Then, applying

all the pairwise reductions, we again get $3 \cdot q^b/2$ samples to be stored in table T^2 and so forth. Hence, if we take

$$\max \{m_{0,\epsilon'}^{\text{LWE}}, 3 \cdot q^b/2\} \quad (29)$$

oracle queries, we are ensured to have enough samples for the Fourier transform. We could thus solve the LWE problem using fewer oracle samples than in Theorem 17 and with a similar time complexity, at the expense of a higher memory complexity (to store multiple samples per table entry).

4.6 Results

We computed the number of operations needed in \mathbb{Z}_q to solve the LWE problem for various values of k when the parameters are chosen according to Regev's cryptosystem [43] and $\epsilon = 0.99$. In this scheme, q is a prime bigger than k^2 and $\sigma = q/(\sqrt{k} \log^2(k) \sqrt{2\pi})$. For our table, we took q to be the smallest prime greater than k^2 . Our results are displayed in Table 1.⁵ To simplify our result, we considered operations over \mathbb{C} to have the same complexity as operations over \mathbb{Z}_q . We also took $C_{\text{FFT}} = 1$ which is the best one can hope to obtain for a FFT. Regarding the noise distribution, we obtained the same results for both $D_{\sigma,q}$ and $\Psi_{\sigma,q}$. If we compare our results with [1, Table1], we see that we are better in all the cases.⁶ This improvement with respect to log likelihood comes from the fact that we do one reduction less in our reduction phase as we recover a full block instead of a single element in \mathbb{Z}_q . This implies that our noise is going to be smaller and, hence, we will need a lower number of queries. However, we still achieve the same asymptotic complexity.

5 Applying our Algorithm to LWR

In this section, we try to apply a similar algorithm to LWR. In the following, we will always consider q to be prime.

Lemma 19. *Let k and $q > p \geq 2$ be positive integers, q prime. Let (\mathbf{a}, c) be a random sample from an LWR oracle $\Lambda_{\mathbf{s},p}$. Then, the “rounding error”, given by $\xi = (p/q)\langle \mathbf{a}, \mathbf{s} \rangle - c$, follows a uniform distribution in a discrete subset of $[-1/2, 1/2]$ with mean zero.*

Furthermore, for $\gamma \in \mathbb{R}_{\neq 0}$,

$$\mathbb{E} [e^{\pm i\xi\gamma}] = \frac{1}{q} \cdot \frac{\sin(\frac{\gamma}{2})}{\sin(\frac{\gamma}{2q})}. \quad (30)$$

⁵ The code used to compute these value is available on our website <http://lasec.epfl.ch/lwe/>

⁶ Albrecht et al. simplified their complexity by considering non-integer a which explains why the difference between our results varies depending on k .

Table 1. We write $\#\mathbb{Z}_q$ for the worst case cost (in operations over \mathbb{Z}_q) of solving Search-LWE for various parameters for the Regev cryptosystem [43] when $\epsilon = 0.99$ according to Theorem 17. We provide also the value of a that minimizes the complexity, the number of queries (m) according to (27), and the number of queries (m) when we apply the LF2 heuristic (29).

k	q	a	$\log(\#\mathbb{Z}_q)$	$\log(m)$	$\log(m)$ for LF2	$\log(\#\mathbb{Z}_q)$ in [1]
64	4 099	19	52.62	43.61	41.01	54.85
80	6 421	20	63.23	53.85	51.18	65.78
96	9 221	21	73.72	63.95	61.98	76.75
112	12 547	21	85.86	75.94	73.20	87.72
128	16 411	22	95.03	84.86	82.05	98.67
160	25 601	23	115.87	105.33	102.46	120.43
224	50 177	24	160.34	149.26	146.32	163.76
256	65 537	25	178.74	167.43	164.43	185.35
384	147 457	26	269.18	257.23	254.17	—
512	262 147	27	357.45	345.03	341.92	—

Proof. We first prove the first part of the lemma. We will prove that for any $\alpha \in [-\frac{q+1}{2}, \dots, \frac{q-1}{2}]$, ξ takes the value α/q with probability $1/q$. We have $p \cdot \langle \mathbf{a}, \mathbf{s} \rangle \equiv \xi q \pmod{q}$. So $\alpha = \xi q = ((p \cdot \langle \mathbf{a}, \mathbf{s} \rangle + (q-1)/2) \bmod q) - (q-1)/2$. Since $\langle \mathbf{a}, \mathbf{s} \rangle$ is uniform in \mathbb{Z}_q (for $\mathbf{s} \neq 0$), α is uniform in $-(q+1)/2, \dots, (q-1)/2$ and has mean zero. Hence, so has ξ .

We now prove the second part of our lemma. Let $X = q \cdot \xi$ be a random variable following a discrete uniform distribution on the set of integers $\{(-q+1)/2, \dots, (q-1)/2\}$. Then, from the characteristic function of X , for any $t \in \mathbb{R}$ we have

$$\mathbb{E}[e^{itX}] = \frac{e^{-it(q-1)/2} - e^{it(q+1)/2}}{q \cdot (1 - e^{it})}. \quad (31)$$

By simple arithmetic, we obtain

$$\mathbb{E}[e^{i\xi\gamma}] = \mathbb{E}[e^{i\gamma q^{-1}X}] = \frac{e^{i\gamma/(2q)} (e^{-i\gamma/2} - e^{i\gamma/2})}{q (1 - e^{i\gamma/q})} = \frac{-\sin(\gamma/2) \cdot 2i}{q (e^{-\gamma i/(2q)} - e^{\gamma i/(2q)})}$$

which gives our result. \square

In our case, q is an odd prime and different from p . Hence, $\mathbb{E}[e^{i\xi\gamma}]$ tends to $\frac{2}{\gamma} \sin(\gamma/2)$ as q grows to infinity. We will be interested in the value $\gamma = 2\pi/p$. Then, for small $p = \{2, 3, 4, 5, \dots\}$, $\mathbb{E}[e^{i\xi\gamma}]$ is $\{0.6366, 0.8270, 0.9003, 0.9355, \dots\}$.

5.1 The LWR-solving Algorithm

From the similarity of the LWR and LWE problems, it should not seem surprising that we would use the same sample reduction and back substition phases, but

we need an alternative ‘‘hypothesis testing phase’’ (which we call *solving phase*) to account for the difference in error distributions.

As for LWE, we choose some $a, b \leq k$ such that $ab \leq k$ and we let $k' = k - (a-1)b$. We will view the reduction phase of our algorithm as producing a series of oracles $\mathcal{B}_{\mathbf{s}, p, \ell}$ for $0 \leq \ell \leq a-1$, where $\mathcal{B}_{\mathbf{s}, p, 0}$ is the original LWR oracle $\Lambda_{\mathbf{s}, p}$. The final oracle $\mathcal{B}_{\mathbf{s}, p, a-1}$ produces samples (\mathbf{a}, c) where \mathbf{a} is non-zero only on the first k' elements.

Solving Phase. We consider the samples from $\mathcal{B}_{\mathbf{s}, p, a-1}$ as belonging to $\mathbb{Z}_q^{k'} \times \mathbb{Z}_p$. We assume we have m such samples and represent them as a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times k'}$ with rows \mathbf{A}_j and a vector $\mathbf{c} \in \mathbb{Z}_p^m$. The corresponding block of k' elements of the secret \mathbf{s} is denoted \mathbf{s}' .

Additionally, we assume that each sample $(\mathbf{a}^{(j)}, c^{(j)})$ from $\mathcal{B}_{\mathbf{s}, p, a-1}$ is the sum of 2^{a-1} samples (or their negation) from the LWR oracle. The ‘noise’ $\langle \mathbf{a}^{(j)}, \mathbf{s}' \rangle \frac{p}{q} - c^{(j)}$ will then correspond to the sum of 2^{a-1} independent ‘‘rounding errors’’ (or their negation) from the original samples.

For $\theta_u := \exp(2\pi i/u)$, we consider the function

$$f_{\text{lwr}}(\mathbf{x}) := \sum_{j=1}^m \mathbf{1}_{\{\mathbf{A}_j = \mathbf{x}\}} \theta_p^{c_j}, \quad \forall \mathbf{x} \in \mathbb{Z}_q^{k'}. \quad (32)$$

The discrete Fourier transform of f_{lwr} is

$$\hat{f}_{\text{lwr}}(\boldsymbol{\alpha}) := \sum_{\mathbf{x} \in \mathbb{Z}_q^{k'}} f_{\text{lwr}}(\mathbf{x}) \theta_q^{-\langle \mathbf{x}, \boldsymbol{\alpha} \rangle} = \sum_{j=1}^m \theta_p^{-((\mathbf{A}_j, \boldsymbol{\alpha}) \frac{p}{q} - c_j)}. \quad (33)$$

In particular, note that

$$\hat{f}_{\text{lwr}}(\mathbf{s}') = \sum_{j=1}^m \theta_p^{-((\mathbf{s}', \boldsymbol{\alpha}) \frac{p}{q} - c_j)} = \sum_{j=1}^m \theta_p^{-(\pm \xi_{j,1} \pm \dots \pm \xi_{j,2^{a-1}})}, \quad (34)$$

where the $\xi_{j,\ell}$ are independent rounding errors from the original LWR samples. Note that it is irrelevant whether the noise has been reduced modulo p , since $\theta_p^{-up} = 1$ for $u \in \mathbb{Z}$.

As for LWE, we can now derive an explicit formula for the number of samples m , which are required to recover \mathbf{s}' with high probability.

Lemma 20. For $q > p \geq 2$, q prime, $\mathbb{E} [\text{Re}(\hat{f}_{\text{lwr}}(\mathbf{s}'))] = m \cdot \left(\frac{1}{q} \cdot \frac{\sin(\frac{\pi}{p})}{\sin(\frac{\pi}{pq})} \right)^{2^{a-1}}$.

Proof. Let ξ be the random variable defined in Lemma 19. Since the original rounding errors are independent, using Lemma 19, we may write

$$\mathbb{E} [\text{Re}(\hat{f}_{\text{lwr}}(\mathbf{s}'))] = m \cdot \text{Re} \left(\mathbb{E} \left[e^{\mp i \xi \frac{2\pi}{p}} \right]^{2^{a-1}} \right) = m \cdot \left(\frac{1}{q} \cdot \frac{\sin(\frac{\pi}{p})}{\sin(\frac{\pi}{pq})} \right)^{2^{a-1}}. \quad (35)$$

□

We need also to bound the values of \widehat{f} when not evaluated at \mathbf{s}' .

Lemma 21. *Let $\boldsymbol{\alpha} \neq \mathbf{s}'$. Then*

$$\mathbb{E} [\operatorname{Re}(\widehat{f}_{\text{lwr}}(\boldsymbol{\alpha}))] \leq m \left(\frac{2}{p} + \frac{1}{p} \cos \left(\frac{\pi}{p} \right) \right)^{2^{a-1}} \leq m \left(\frac{3}{p} \right)^{2^{a-1}}.$$

Proof. Like in the previous lemma, we can write, for \mathbf{a} uniformly distributed,

$$\mathbb{E} [\operatorname{Re}(\widehat{f}_{\text{lwr}}(\boldsymbol{\alpha}))] = m \cdot \operatorname{Re} \left(\mathbb{E} \left[e^{\mp i(2\pi\langle \mathbf{a}, \boldsymbol{\alpha} \rangle / q - 2\pi c / p)} \right]^{2^{a-1}} \right). \quad (36)$$

However, unlike in the LWE case, we cannot use the independence of \mathbf{a} and the noise to obtain a zero expected value. This occurs because the errors are computed deterministically from the vectors \mathbf{a} in LWR. In fact, experiments showed that the error is strongly correlated to \mathbf{a} and that the expected value is not zero. Thus, we will instead bound this expected value. To do this, we write

$$\mathbb{E} \left[e^{\mp i(2\pi\langle \mathbf{a}, \boldsymbol{\alpha} \rangle / q - 2\pi c / p)} \right] = \mathbb{E} \left[\cos \left(\frac{2\pi\langle \mathbf{a}, \boldsymbol{\alpha} \rangle}{q} - \frac{2\pi c}{p} \right) \right] \pm i \cdot \mathbb{E} \left[\sin \left(-\frac{2\pi\langle \mathbf{a}, \boldsymbol{\alpha} \rangle}{q} + \frac{2\pi c}{p} \right) \right]$$

and we bound both the sine and the cosine term.

- We first show that the contribution of the sine is zero, i.e., that for $\boldsymbol{\alpha} \neq \mathbf{s}'$ fixed,⁷

$$\mathbb{E} [\sin(2\pi\langle \mathbf{a}, \boldsymbol{\alpha} \rangle / q - 2\pi c / p)] = 0. \quad (37)$$

Let $w(\mathbf{a}) := \sin(2\pi\langle \mathbf{a}, \boldsymbol{\alpha} \rangle / q - 2\pi \lceil \langle \mathbf{a}, \mathbf{s}' \rangle (p/q) \rceil / p)$. First, note that for $\mathbf{a} = \mathbf{0}$, $c = 0$. For $\mathbf{a} \neq \mathbf{0}$, the contribution in the expected value is $w(\mathbf{a})$. We have

$$\begin{aligned} w(-\mathbf{a}) &= \sin(2\pi\langle -\mathbf{a}, \boldsymbol{\alpha} \rangle / q - 2\pi \lceil \langle -\mathbf{a}, \mathbf{s}' \rangle (p/q) \rceil / p) \\ &= \sin(-2\pi\langle \mathbf{a}, \boldsymbol{\alpha} \rangle / q - 2\pi \lceil -\langle \mathbf{a}, \mathbf{s}' \rangle (p/q) \rceil / p) = -w(\mathbf{a}). \end{aligned}$$

Since q is odd, $-\mathbf{a} \neq \mathbf{a}$ and, thus, in the expected value, the contribution of any $\mathbf{a} \neq 0$ is cancelled. Hence, the result.

- For the cosine, as in Lemma 15, we let $\mathbf{y} = \boldsymbol{\alpha} - \mathbf{s}' \in \mathbb{Z}_q^{k'}$. We get,

$$\begin{aligned} \cos \left(\frac{2\pi\langle \mathbf{a}, \boldsymbol{\alpha} \rangle}{q} - \frac{2\pi c}{p} \right) &= \cos \left(\frac{2\pi\langle \mathbf{a}, \mathbf{y} \rangle}{q} + \frac{2\pi(\langle \mathbf{a}, \mathbf{s}' \rangle p/q - c)}{p} \right) \\ &= \cos \left(\frac{2\pi\langle \mathbf{a}, \mathbf{y} \rangle}{q} + \frac{2\pi\xi}{p} \right), \end{aligned} \quad (38)$$

where $\xi \in [-1/2, 1/2]$ is the rounding error from Lemma 19. We are looking for an upper-bound and, hence, we assume that $\xi \in [-1/2, 1/2]$ will always

⁷ This is where the round function instead of the floor function in the definition of LWR becomes handy.

be such that $\cos(2\pi\langle \mathbf{a}, \mathbf{y} \rangle/q + 2\pi\xi/p)$ is maximized. Figure 1 might help with the reading. We divide the circle into sets of the form

$$\mathcal{S}_\ell := \left[\frac{\ell\pi}{p}, \frac{(\ell+1)\pi}{p} \right] \cup \left[\frac{-\ell\pi}{p}, \frac{-(\ell+1)\pi}{p} \right], \quad \ell \in [0, p-1].$$

Note that this covers the whole circle. The hashed surface in Figure 1 is such a set.

When $2\pi\langle \mathbf{a}, \mathbf{y} \rangle/q \in \mathcal{S}_\ell$ for $\ell \neq 0$, we upper-bound (38) by $\cos((\ell-1)\pi/p)$ (the bold line in Figure 1). Indeed, $|2\pi\xi/p| \leq \pi/p$. When $2\pi\langle \mathbf{a}, \mathbf{y} \rangle/q \in \mathcal{S}_0$, we upper-bound (38) by $\cos(0) = 1$.

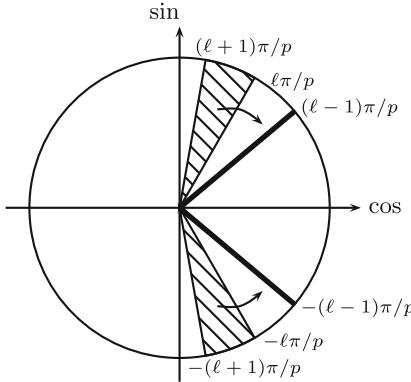


Fig. 1. Figure for the proof of Lemma 21

Note that $\Pr[2\pi\langle \mathbf{a}, \mathbf{y} \rangle/q \in \mathcal{S}_\ell] = 1/p$ since $\langle \mathbf{a}, \mathbf{y} \rangle$ is uniformly distributed in \mathbb{Z}_q and $p \leq q$. Hence,

$$\begin{aligned} \mathbb{E} [\cos(2\pi\langle \mathbf{a}, \mathbf{y} \rangle + 2\pi\xi/p)] &\leq \frac{1}{p} + \frac{1}{p} \sum_{\ell=1}^{p-1} \cos\left(\frac{(\ell-1)\pi}{p}\right) \\ &= \frac{1}{p} + \frac{1}{p} \cos(0) - \frac{1}{p} \cos\left(\frac{(p-1)\pi}{p}\right) = \frac{2}{p} + \frac{1}{p} \cos\left(\frac{\pi}{p}\right) \leq \frac{3}{p}. \end{aligned} \quad (39)$$

Plugging the values of the sine and the upper-bound for the cosine in (36) finishes the proof. \square

Lemma 22. *When $q > p \geq 4$ and q is prime, $\arg \max_{\boldsymbol{\alpha}} \text{Re}(\widehat{f}_{\text{lwr}}(\boldsymbol{\alpha})) = \mathbf{s}'$ with probability greater than*

$$1 - q^{k'} \cdot \exp\left(-\frac{m}{8} \cdot \left(\left(\frac{1}{q} \cdot \frac{\sin(\frac{\pi}{p})}{\sin(\frac{\pi}{pq})}\right)^{2^{a-1}} - \left(\frac{3}{p}\right)^{2^{a-1}}\right)^2\right).$$

Proof. We first want the probability that $\text{Re}(\widehat{f}(\mathbf{x})) \geq \text{Re}(\widehat{f}(\mathbf{s}'))$ for some fixed vector $\mathbf{x} \in \mathbb{Z}_q^{k'}, \mathbf{x} \neq \mathbf{s}'$. Applying the same heuristic argument as for LWE, we consider X_1, X_2, \dots, X_m to be independent random variables with $X_j = u_j - v_j$, where

$$u_j = \text{Re} \left(\theta_p^{-\langle (\mathbf{A}_j, \mathbf{s}') \rangle_q^p - c_j} \right) \quad \text{and} \quad v_j = \text{Re} \left(\theta_p^{-\langle (\mathbf{A}_j, \mathbf{x}) \rangle_q^p - c_j} \right). \quad (40)$$

Note that $X_j \in [-2, 2]$ for all j . Furthermore, let $X = \sum_{j=1}^m X_j$. Using Lemmas 20 and 21, we get that

$$\mathbb{E}[X] \geq m \cdot \left(\left(\frac{1}{q} \cdot \frac{\sin(\frac{\pi}{p})}{\sin(\frac{\pi}{pq})} \right)^{2^{a-1}} - \left(\frac{3}{p} \right)^{2^{a-1}} \right) \geq 0. \quad (41)$$

We will again bound the probability that $X \leq 0$ using Hoeffding's inequality. Let $t = \mathbb{E}[X] > 0$. Then,

$$\begin{aligned} \Pr[X \leq 0] &= \Pr[(X - \mathbb{E}[X]) \leq -\mathbb{E}[X]] \leq \exp \left(\frac{-2(\mathbb{E}[X])^2}{16m} \right) \\ &\leq \exp \left(-\frac{m}{8} \cdot \left(\left(\frac{1}{q} \cdot \frac{\sin(\frac{\pi}{p})}{\sin(\frac{\pi}{pq})} \right)^{2^{a-1}} - \left(\frac{3}{p} \right)^{2^{a-1}} \right)^2 \right). \end{aligned} \quad (42)$$

The final result follows by applying a union bound over all possible values of \mathbf{x} . \square

As for LWE, we may now deduce the number m of reduced samples that are required to recover a block \mathbf{s}' .

Theorem 23. *Let k and $q > p \geq 4$ be positive integers, q prime, and $\Lambda_{\mathbf{s},p}$ be an LWR oracle, where $\mathbf{s} \in \mathbb{Z}_q^k$. Let $a \in \mathbb{Z}$ with $1 \leq a \leq k$, let b be such that $ab \leq k$, and let $k' = k - (a-1)b$. Let $\mathcal{B}_{\mathbf{s},p,a-1}$ be the oracle returning samples (\mathbf{a}_i, c_i) where the \mathbf{a}_i are zero for all but the first k' elements. Denote the vector consisting of the first k' elements of \mathbf{s} as \mathbf{s}' . Fix an $\epsilon \in (0, 1)$. Then, the number of samples m from $\mathcal{B}_{\mathbf{s},p,a-1}$, which are required such that we fail to recover the secret block \mathbf{s}' with probability at most ϵ satisfies*

$$m^{LWR} \geq 8 \cdot k' \cdot \log \left(\frac{q}{\epsilon} \right) \cdot \left(\left(\frac{1}{q} \cdot \frac{\sin(\frac{\pi}{p})}{\sin(\frac{\pi}{pq})} \right)^{2^{a-1}} - \left(\frac{3}{p} \right)^{2^{a-1}} \right)^{-2}.$$

Furthermore, recovering \mathbf{s}' in the solving phase (the FFT phase) requires $2m^{LWR} + C_{\text{FFT}} \cdot k' \cdot q^{k'} \cdot \log q$ operations in \mathbb{C} , as well as storage for $q^{k'}$ complex numbers.

We now summarize the complexity of our algorithm in the following theorem (the proof of which is analogous to the proof of Theorem 17).

Theorem 24 (Complexity of SEARCH-LWR). Let $k, q > p \geq 4$ be positive integers, q prime, and $\Pi_{\mathbf{s}, \chi}$ be an LWE oracle, where $\mathbf{s} \in \mathbb{Z}_q^k$. Let $a, b \in \mathbb{Z}_q$ be such that $a \cdot b = k$. For $0 \leq j \leq a - 1$, let

$$m_{j,\epsilon}^{LWR} := 8 \cdot b \cdot \log\left(\frac{q}{\epsilon}\right) \cdot \left(\left(\frac{1}{q} \cdot \frac{\sin(\frac{\pi}{p})}{\sin(\frac{\pi}{pq})} \right)^{2^{a-1-j}} - \left(\frac{3}{p} \right)^{2^{a-1-j}} \right)^{-2}.$$

Let $0 < \epsilon < 1$ be a targeted success rate and define $\epsilon' := (1 - \epsilon)/a$. The (time, memory and query) complexities to recover the LWR secret \mathbf{s} with probability ϵ are the same as in Theorem 17 where we replace $m_{j,\epsilon}^{LWE}$ by $m_{j,\epsilon}^{LWR}$.

5.2 Results

The current hardness results for LWR require either a parameter q exponential in k or a bound m on the number of oracle samples that an adversary may query. It is an open problem ([3]) to assess the hardness of LWR with polynomial parameters when the adversary has no sample limit. In such a case, for $a = O(\log k)$ and $b = \lceil k/a \rceil$, our algorithm would solve LWR in time $2^{O(k)}$, as for LWE.

However, the bound on the number of oracle samples in Theorem 7 is much lower than the amount of samples required by our algorithm. Using an idea from Lyubashevsky [37] we can generate additional samples with higher noise (see Theorem 18). Yet, even this method requires at least $k^{1+\epsilon}$ samples for $\epsilon \geq (\log q + 1)/\log k$, which is incompatible with the constraints of Theorem 7, for a q polynomial in k .

in [3, Corollary 4.2], two types of parameters are proposed: parameters minimizing the Modulus/Error ratio (a) and parameters maximizing efficiency (b). For completeness, we show in Table 2 the complexity of our algorithm applied to these parameters. More precisely, we took for the underlying LWE problem

Table 2. Worst case cost (in operations over \mathbb{Z}_q) of solving Search-LWR for various parameters for the Regev cryptosystem [43] when $\epsilon = 0.99$ according to Theorem 24. We provide also the value of a that minimizes the complexity, the number of queries (m) according to (27).

k	q	p	a	$\log(\#\mathbb{Z}_q)$	$\log(m)$	type
64	383 056 211	733	23	92.20	82.80	(a)
80	1 492 443 083	1 151	25	110.91	101.11	(a)
96	$\approx 2^{32}$	1 663	26	132.26	122.15	(a)
112	$\approx 2^{33}$	2 287	28	148.08	137.68	(a)
128	$\approx 2^{34}$	3 023	29	167.52	156.87	(a)
64	9 461	13	12	81.61	72.90	(b)
80	14 867	13	12	103.89	94.86	(b)
96	21 611	13	12	126.97	117.66	(b)
112	29 717	13	13	140.21	130.60	(b)
128	39 241	13	13	162.63	152.84	(b)

Regev's parameters and *ignored the constrains on the number of samples*. For the type (a) parameters, we took

$$\sigma = \frac{k^2}{\sqrt{k} \log^2(k) \sqrt{2\pi}} \quad q = \text{nextprime}(\lceil (2\sigma k)^3 \rceil) \quad p = \text{nextprime}(\lceil \sqrt[3]{q} \rceil)$$

and for the type (b) parameters

$$\sigma = \frac{k^2}{\sqrt{k} \log^2(k) \sqrt{2\pi}} \quad p = 13 \quad q = \text{nextprime}(\lceil 2\sigma kp \rceil).$$

Table 2 shows that the parameters proposed in [3] seem secure even if we remove the constrain on the number of samples as the complexities are still quite high.

6 Conclusion

To summarize, we propose an algorithm which is currently the best algorithm for solving the LWE problem. Our algorithm uses Fourier transforms and we propose a careful analysis of the rounded Gaussian distribution which can be of independent interest. In particular, we study its variance and the expected value of its cosine. We also adapt our algorithm to the LWR problem when q is prime. This algorithm is the first LWR-solving algorithm.

Further work includes the study of the Ring variants of LWE and LWR [8, 38] and the study of variants of LWE, e.g., when the secret follows a non-uniform distribution (like in [2]) or when the noise follows a non Gaussian distribution. It would also be interesting to see if our LWR algorithm can be extended for q non prime.

Acknowledgments. We are grateful to Dimitar Jetchev and Adeline Langlois for helpful discussions and pointers. We thank the Eurocrypt 2015 reviewers for their fruitful comments.

A Continuous Fourier Transforms

We use the following definition for continuous Fourier Transforms. The continuous Fourier transform (FT) of a function $f: \mathbb{R} \rightarrow \mathbb{C}$ is a function $\mathcal{F}(f): \mathbb{R} \rightarrow \mathbb{C}$ defined as

$$\mathcal{F}(f)(\chi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \chi x} dx. \quad (43)$$

We will use the following well-known properties.

Linearity.

$$\mathcal{F}(f(x) + g(x))(\chi) = (\mathcal{F}(f) + \mathcal{F}(g))(\chi). \quad (44)$$

Translation.

$$\mathcal{F}(f(x-y))(\chi) = e^{-i2\pi y\chi} \mathcal{F}(f)(\chi) . \quad (45)$$

Convolution.

$$\mathcal{F}(f(x)g(x))(\chi) = (\mathcal{F}(f(x)) * \mathcal{F}(g(x)))(\chi) , \quad (46)$$

where $*$ denotes the convolution operator which is defined as

$$(u * v)(x) := \int_{-\infty}^{\infty} u(y)v(x-y) dy .$$

Integration.

$$\mathcal{F}\left(\int_{-\infty}^x f(\tau) d\tau\right)(\chi) = \frac{1}{2i\pi\chi} \mathcal{F}(f)(\chi) + \frac{1}{2} \mathcal{F}(f)(0)\delta(\chi) , \quad (47)$$

where δ is the Dirac delta distribution. We will use the following property of the Dirac delta.

$$\int_{-\infty}^{\infty} f(\tau)\delta(\tau - \ell) d\tau = f(\ell) .$$

We refer the reader to, e.g., [24, 45, 47] for more information about the Dirac delta distribution and its derivatives or, e.g. [14] for a more engineering approach.

We will also use the Poisson summation formula.

Lemma 25. *Poisson summation formula (see, e.g., [46])* Let $f(x): \mathbb{R} \rightarrow \mathbb{C}$ be a function in the Schwartz space⁸ and $\mathcal{F}(f)$ its continuous Fourier transform then

$$\sum_{\ell=-\infty}^{\infty} f(\ell) = \sum_{\chi=-\infty}^{\infty} \mathcal{F}(f)(\chi) . \quad (48)$$

Useful Fourier Transforms.

$$\mathcal{F}\left(\frac{1}{\sigma\sqrt{2\pi}}e^{-x^2/(2\sigma^2)}\right)(\chi) = e^{-2\pi^2\sigma^2\chi^2} . \quad (49)$$

Let $\gamma \in \mathbb{R}$. Then

$$\mathcal{F}(\cos(\alpha x))(\chi) = \frac{1}{2} \left(\delta\left(\chi - \frac{\gamma}{2\pi}\right) + \delta\left(\chi + \frac{\gamma}{2\pi}\right) \right) , \quad (50)$$

where δ is the Dirac delta distribution.

⁸ A function $f(x)$ is in the Schwartz space if $\forall \alpha, \beta \in \mathbb{N}, \exists C_{\alpha, \beta}$ such that $\sup|x^\alpha \partial_x^\beta f(x)| \leq C_{\alpha, \beta}$. A function in C^∞ with compact support is in the Schwartz space.

References

1. Albrecht, M.R., Cid, C., Faugère, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. In: Designs, Codes and Cryptography, pp. 1–30 (2013)
2. Albrecht, M.R., Faugère, J.-C., Fitzpatrick, R., Perret, L.: Lazy Modulus Switching for the BKW Algorithm on LWE. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 429–445. Springer, Heidelberg (2014)
3. Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited - new reduction, properties and applications. In: Canetti and Garay [19], pp. 57–74
4. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009)
5. Arora, S., Ge, R.: New Algorithms for Learning in Presence of Errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 403–415. Springer, Heidelberg (2011)
6. Arthur Pewsey, Markus Neuhäuser, G.D.R.: Circular statistics in R. Oxford University Press (2013)
7. Banaszczky, W.: New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen* **296**(1), 625–635 (1993)
8. 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)
9. Becker, A., Gama, N., Joux, A.: A sieve algorithm based on overlattices. *LMS Journal of Computation and Mathematics* **17**, 49–70 (1 2014)
10. Bernstein, D.J., Lange, T.: Never Trust a Bunny. In: Hoepman, J.-H., Verbauwheide, I. (eds.) RFIDSec 2012. LNCS, vol. 7739, pp. 137–148. Springer, Heidelberg (2013)
11. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM* **50**(4), 506–519 (2003)
12. Bogos, S., Tramèr, F., Vaudenay, S.: On Solving LPN using BKW and Variants. *Cryptology ePrint Archive*, Report 2015/049 (2015). <http://eprint.iacr.org/>
13. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key Homomorphic PRFs and Their Applications. In: Canetti and Garay [19], pp. 410–428
14. Bracewell, R.N., Bracewell, R.: The Fourier transform and its applications, vol. 31999. McGraw-Hill, New York (1986)
15. Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012)
16. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, June 1–4, 2013. pp. 575–584. ACM (2013)
17. Brakerski, Z., Vaikuntanathan, V.: Efficient Fully Homomorphic Encryption from (Standard) LWE. In: Ostrovsky, R. (ed.) IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22–25, 2011. pp. 97–106. IEEE (2011)
18. Buhler, J., Shokrollahi, M.A., Stemmann, V.: Fast and precise Fourier transforms. *IEEE Transactions on Information Theory* **46**(1), 213–228 (2000)

19. Canetti, R., Garay, J.A. (eds.): Advances in Cryptology - CRYPTO 2013–33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2013. Proceedings, Part I, Lecture Notes in Computer Science, vol. 8042. Springer (2013)
20. Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.): Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, APPROX 2005 and RANDOM 2005, Lecture Notes in Computer Science, vol. 3624. Springer (2005)
21. 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)
22. Fossorier, M.P.C., Mihaljević, M.J., Imai, H., Cui, Y., Matsuura, K.: An Algorithm for Solving the LPN Problem and Its Application to Security Evaluation of the HB Protocols for RFID Authentication. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 48–62. Springer, Heidelberg (2006)
23. Gama, N., Nguyen, P.Q., Regev, O.: Lattice Enumeration Using Extreme Pruning. In: Gilbert [28], pp. 257–278
24. Gelfand, I.M., Shilov, G.: Generalized functions. Vol. 1. Properties and operations (1964)
25. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009). <http://crypto.stanford.edu/craig>
26. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.) Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17–20, 2008. pp. 197–206. ACM (2008)
27. Gentry, C., Sahai, A., Waters, B.: Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In: Canetti and Garay [19], pp. 75–92
28. Gilbert, H. (ed.): Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6110. Springer (2010)
29. Goldwasser, S., Kalai, Y.T., Peikert, C., Vaikuntanathan, V.: Robustness of the Learning with Errors Assumption. In: Yao, A.C. (ed.) Proceedings of the Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5–7, 2010, pp. 230–240. Tsinghua University Press (2010)
30. 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)
31. Hanrot, G., Pujol, X., Stehlé, D.: Analyzing Blockwise Lattice Algorithms Using Dynamical Systems. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 447–464. Springer, Heidelberg (2011)
32. Heyse, S., Kiltz, E., Lyubashevsky, V., Paar, C., Pietrzak, K.: Lapin: An Efficient Authentication Protocol Based on Ring-LPN. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 346–365. Springer, Heidelberg (2012)
33. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American statistical association **58**(301), 13–30 (1963)
34. Impagliazzo, R., Zuckerman, D.: How to Recycle Random Bits. In: FOCS. pp. 248–253. IEEE Computer Society (1989)
35. Levieil, É., Fouque, P.-A.: An Improved LPN Algorithm. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 348–359. Springer, Heidelberg (2006)

36. 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)
37. Lyubashevsky, V.: The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In: Chekuri et al. [20], pp. 378–389
38. Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings. In: Gilbert [28], pp. 1–23
39. Mardia, K., Jupp, P.: Directional Statistics. Wiley, Wiley Series in Probability and Statistics (2009)
40. Nguyen, P.Q.: Lattice Reduction Algorithms: Theory and Practice. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 2–6. Springer, Heidelberg (2011)
41. Nguyen, P.Q., Stehlé, D.: Low-dimensional lattice basis reduction revisited. ACM Transactions on Algorithms 5(4) (2009)
42. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009, pp. 333–342. ACM (2009)
43. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM 56(6) (2009)
44. Regev, O.: The learning with errors problem (invited survey). In: IEEE Conference on Computational Complexity. pp. 191–204. IEEE Computer Society (2010)
45. Rudin, W.: Functional analysis. McGraw-Hill Inc, New York (1991)
46. Eickhoff, J.: Introduction. In: Eickhoff, J. (ed.) Onboard Computers, Onboard Software and Satellite Operations. SAT, vol. 1, pp. 3–6. Springer, Heidelberg (2012)
47. Strichartz, R.S.: A guide to distribution theory and Fourier transforms. World Scientific (2003)

On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes

Alexander May^(✉) and Ilya Ozerov

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

Abstract. We propose a new decoding algorithm for random binary linear codes. The so-called information set decoding algorithm of Prange (1962) achieves worst-case complexity $2^{0.121n}$. In the late 80s, Stern proposed a sort-and-match version for Prange's algorithm, on which all variants of the currently best known decoding algorithms are build. The fastest algorithm of Becker, Joux, May and Meurer (2012) achieves running time $2^{0.102n}$ in the full distance decoding setting and $2^{0.0494n}$ with half (bounded) distance decoding.

In this work we point out that the sort-and-match routine in Stern's algorithm is carried out in a non-optimal way, since the matching is done in a two step manner to realize an *approximate matching* up to a small number of error coordinates. Our observation is that such an approximate matching can be done by a variant of the so-called High Dimensional Nearest Neighbor Problem. Namely, out of two lists with entries from \mathbb{F}_2^m we have to find a pair with closest Hamming distance. We develop a new algorithm for this problem with sub-quadratic complexity which might be of independent interest in other contexts.

Using our algorithm for full distance decoding improves Stern's complexity from $2^{0.117n}$ to $2^{0.114n}$. Since the techniques of Becker et al apply for our algorithm as well, we eventually obtain the fastest decoding algorithm for binary linear codes with complexity $2^{0.097n}$. In the half distance decoding scenario, we obtain a complexity of $2^{0.0473n}$.

Keywords: Linear codes · Nearest neighbor problem · Approximate matching · Meet-in-the-middle

1 Introduction

The NP-hard decoding problem for random linear codes is one of the most fundamental combinatorial problems in coding and complexity theory. Due to its purely combinatorial structure it is the major source for constructing cryptographic hardness assumptions that retain their hardness even in the presence of

A. May—Supported by DFG as part of GRK 1817 Ubicrypt and SPP 1736 Big Data.

I. Ozerov—Supported by DFG as part of SPP 1307 Algorithm Engineering.

quantum computers. Almost all code-based cryptosystems, such as the McEliece encryption scheme [15], rely on the fact that random linear codes are hard to decode.

The way cryptographers usually embed a trapdoor into code-based constructions is that they start with some structured code C , which allows for efficient decoding, and then use a linear transformation to obtain a scrambled code C' . The cryptographic transformation has to ensure that the scrambled code C' is indistinguishable from a purely random code. Unless somebody is able to unscramble the code, a cryptanalyst faces the problem of decoding a random linear code. Hence, for choosing appropriate security parameters for a cryptographic scheme it is crucial to know the best performance of generic decoding algorithms for linear codes.

Also closely related to random linear codes is the so-called Learning Parity with Noise Problem (LPN) that is frequently used in cryptography [9, 12]. In LPN, one directly starts with a generator matrix that defines a random linear code C and the LPN search problem is a decoding problem on C . Cryptographers usually prefer decision versions of hard problems for proving security of their schemes. However, for LPN there is a reduction from the decision to the search version that directly links the cryptographic hardness of the underlying schemes to the task of decoding random linear codes.

The Learning with Errors Problem (LWE) that was introduced for cryptographic purposes in the work of Regev [19, 20] can be seen as a generalization of LPN to codes defined over larger fields. LWE is closely related to well-studied problems in learning theory, and it proved to be a fruitful source within the last decade for many cryptographic constructions that provide new functionalities [5, 7, 16]. Although we focus in this work on random linear codes over \mathbb{F}_2 , we do not see any obstacles in transferring our techniques to larger finite fields \mathbb{F}_q as this was done in [17]. Surely, our techniques will also lead to some improvement for arbitrary fields, but we believe that our improvements are best tailored to \mathbb{F}_2 , easiest to explain for the binary field, and we feel that binary codes define the most fundamental and widely applied class of linear codes.

Let us define some basics of linear codes and review the progress that decoding algorithms underwent. A (random) binary linear code C is a (random) k -dimensional subspace of \mathbb{F}_2^n . Therefore, a code defines a mapping $\mathbb{F}_2^k \rightarrow \mathbb{F}_2^n$ that maps a message $\mathbf{m} \in \mathbb{F}_2^k$ to a codeword $\mathbf{c} \in \mathbb{F}_2^n$. On a noisy channel, a receiver gets an erroneous version $\mathbf{x} = \mathbf{c} + \mathbf{e}$ for some error vector $\mathbf{e} \in \mathbb{F}_2^n$. The decoding problem now asks for finding \mathbf{e} , which in turn enables to reconstruct \mathbf{c} and \mathbf{m} . Usually, we assume that during transmission of \mathbf{c} not too many errors occurred, such that \mathbf{e} has a small Hamming weight $\text{wt}(\mathbf{e})$ and \mathbf{c} is the closest codeword to \mathbf{x} . This defines the search space of \mathbf{e} , which is closely linked to the distance d of C .

So naturally, the running time $T(n, k, d)$ of a decoding algorithm is a function of all code parameters n, k and d . However, we know that asymptotically random linear codes reach the Gilbert-Varshamov bound $\frac{k}{n} \leq 1 - H(\frac{d}{n})$, where $H(\cdot)$ is the binary entropy function (see e.g. [22]). In the *full distance decoding* setting we

are looking for an error vector \mathbf{e} s.t. $\text{wt}(\mathbf{e}) \leq d$, whereas in *half distance decoding* we have $\text{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$. Thus, in both cases we can upper bound the running time by a function $T(n, k)$ of n and k only. When we speak of worst-case running time, we maximize $T(n, k)$ for all k where the maximum is obtained for code rates $\frac{k}{n}$ near $\frac{1}{2}$. Usually, it suffices to compare worst-case complexities since all known decoding algorithms with running time T, T' and worst-case complexity $T(n) < T'(n)$ satisfy $T(n, k) < T'(n, k)$ for all k .

The simplest algorithm is to enumerate naively over \mathbf{e} 's search space and check whether $\mathbf{x} + \mathbf{e} \in C$. However, it was already noticed in 1962 by Prange [18] that the search space for \mathbf{e} can considerably be lowered by applying simple linear algebra. Prange's algorithm consists of an enumeration step with exponential complexity and some Gaussian elimination step with only polynomial complexity. The worst-case complexity of Prange's algorithm is $2^{0.121n}$ in the full distance decoding case and $2^{0.0576n}$ with half distance decoding.

In 1989, Stern [21] noticed that Prange's enumeration step can be accelerated by enumerating two lists \mathbf{L}, \mathbf{R} within half of the search space, a typical time-memory trade-off. The lists \mathbf{L}, \mathbf{R} are then sorted and one looks for a matching pair. This is a standard trick for many combinatorial search problems and is usually called a sort-and-match or Meet-in-the-middle approach in the literature. Stern's algorithm however is unable to directly realize an approximate matching of lists, where one wants to find a pair of vectors from $\mathbf{L} \times \mathbf{R}$ with small Hamming distance. In Stern's algorithm this is solved by a non-optimal two-step approach, where one first matches vector pairs *exactly* on some portion of the coordinates, and then in a second step checks whether any of these pairs has the desired distance on all coordinates. Stern's algorithm led to a running time improvement to $2^{0.117n}$ (full distance) and $2^{0.0557n}$ (half distance), respectively.

Our Contribution: In this work, we propose a different type of matching algorithm for Stern's algorithm that directly recovers a pair from $\mathbf{L} \times \mathbf{R}$ with small Hamming distance. Fortunately, this problem is well-known in different variants in many fields of computer science as the High Dimensional Nearest Neighbor Problem [6, 8, 23] or the Bichromatic Closest Pair Problem [2]. The best bounds that are known for the Hamming metric are due to Dubiner [6] and Valiant [24].

However, we were not able to apply Dubiner's algorithm to our decoding problem, since we have a bounded vector size, which is referred to as the *limited amount of data* case in [6]. Although it is stated in Dubiner's work [6] that his algorithm might also be applicable to the *limited* case, the analysis is only done for the *unlimited amount of data* case. The algorithm of Valiant [24] is only optimal in special cases (i.e. large Hamming distances) and unfortunately doesn't apply to our problem either. Thus we decided to give an own algorithmic solution, which gives us the required flexibility for choosing parameters that are tailored to the decoding setting.

We provide a different and quite general algorithm for finding a pair of (limited or unlimited size) vectors in two lists that fulfill some consistency criterion, like e.g. in our case being close in some distance metric. The way we solve this

problem is by checking parts of the vectors locally, and thus sorting out vector pairs that violate consistency locally, since these pairs are highly unlikely to fulfill consistency globally. In our special case, where $|\mathbf{L}| = |\mathbf{R}|$ and where we want to find the closest pair of vectors from \mathbb{F}_2^m with Hamming distance γm , $\gamma \leq \frac{1}{2}$, we obtain an algorithm that approaches sub-quadratic complexity $|\mathbf{L}|^{\frac{1}{1-\gamma}}$. In our analysis in Sections 4 and 5 we propose an algorithm for bounded vector size and show that in the *unlimited amount of data* case (which is not important for the decoding problem) our algorithm approaches Dubiner's bound.

Using this matching algorithm in the full distance decoding setting directly leads to an improved decoding algorithm for random binary linear codes with complexity $2^{0.114n}$, as opposed to Stern's complexity $2^{0.117n}$. In 2011, Stern's algorithm was improved by Bernstein, Lange and Peters to $2^{0.116n}$. Then, using recent techniques from improving subset sum algorithms [3, 11], May, Meurer, Thomae [14] and Becker, Joux, May, Meurer [4] further improved the running time to $2^{0.102n}$. Fortunately, these techniques directly apply to our algorithm as well and result in a new worst-case running time as small as $2^{0.097n}$. We obtain similar results in the case of half distance decoding.

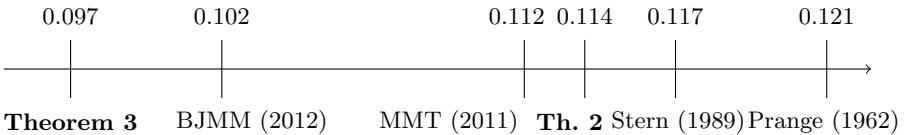


Fig. 1. History of Information Set Decoding: full distance decoding (FDD)

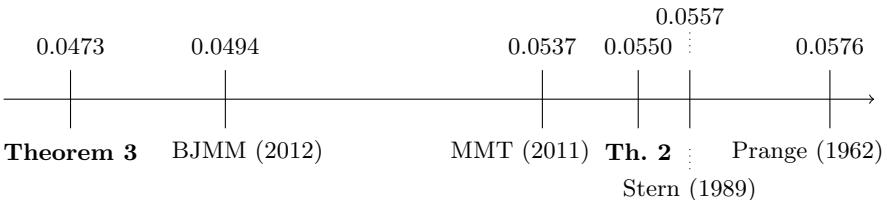


Fig. 2. History of Information Set Decoding: half distance decoding (HDD)

The paper is organized as follows. In Section 2, we elaborate a bit more on previous work and explain the basic idea of our matching approach. This leads to a sort-and-match decoding algorithm that we describe and analyze in Section 3, including the application of the improvement from Becker et al [4].

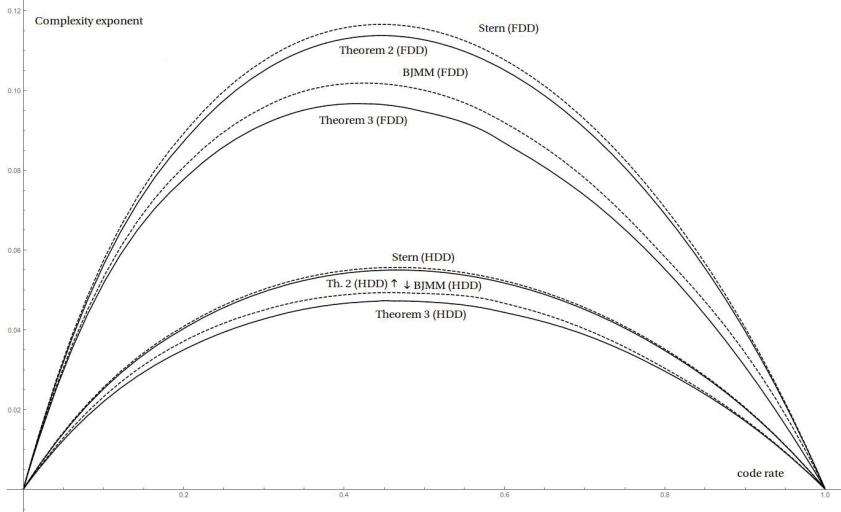


Fig. 3. Stern, Th. 2, BJMM and Th. 3 for BDD/FDD and all code rates k/n

Our decoding algorithm calls a new matching subroutine that finds a pair of vectors with minimal Hamming distance in two lists. We describe this matching procedure in Section 4.

2 Previous Work – Information Set Decoding

A binary linear code C is a k -dimensional subspace of \mathbb{F}_2^n . Thus, C is generated by a matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ that defines a linear mapping $\mathbb{F}_2^k \rightarrow \mathbb{F}_2^n$. If \mathbf{G} 's entries are chosen independently and uniformly at random from $\mathbb{F}_2^{k \times n}$ with the restriction that \mathbf{G} has rank k , then we call C a random binary linear code. The distance d of C is defined by the minimum Hamming distance of two different codewords in C .

Let $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ be a basis of the kernel of C . If C is random then it is not hard to see that the entries of \mathbf{H} are also independently and uniformly at random distributed in \mathbb{F}_2 . Therefore, we have $\mathbf{H}\mathbf{c}^t = \mathbf{0}$ for every $\mathbf{c} \in C$. For simplicity, we omit from now on all transposition of vectors and write \mathbf{c} instead of \mathbf{c}^t .

For every erroneous codeword $\mathbf{x} = \mathbf{c} + \mathbf{e}$ with error vector \mathbf{e} , we obtain $\mathbf{Hx} = \mathbf{He}$ by linearity. We call $\mathbf{s} := \mathbf{Hx} \in \mathbb{F}_2^{n-k}$ the *syndrome* of a message \mathbf{x} . In order to decode \mathbf{x} , it suffices to find a low weight vector \mathbf{e} such that $\mathbf{He} = \mathbf{s}$. Once \mathbf{e} is found, we can simply recover \mathbf{c} from \mathbf{x} . This process is called syndrome decoding, and the problem is known to be NP-hard.

In the case of *full distance decoding* (FDD), we receive an arbitrary point $\mathbf{x} \in \mathbb{F}_2^n$ and want to decode to the closest codeword in the Hamming metric.

We want to argue that there is always a codeword within (roughly) Hamming distance d . Therefore, we observe that the syndrome equation $\mathbf{H}\mathbf{e} = \mathbf{s}$ is solvable as long as the search space for \mathbf{e} roughly equals 2^{n-k} . Hence, for weight- d vectors $\mathbf{e} \in \mathbb{F}_2^n$ we obtain $\binom{n}{d} \approx 2^{H(\frac{d}{n})n} \approx 2^{n-k}$, where $H(\cdot)$ is the binary entropy function. This implies $H(\frac{d}{n}) \approx 1 - \frac{k}{n}$, a relation that is known as the Gilbert-Varshamov bound. Moreover, it is well-known that random codes asymptotically reach the Gilbert-Varshamov bound [22]. This implies that for every \mathbf{x} we can always expect to find a closest codeword within distance d .

In the other case of *half distance decoding* (HDD), we obtain the promise that the error vector is within the error correction distance, i.e. $\mathbf{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$, which is for example the case in cryptographic settings, where an error is added artificially by e.g. the encryption of a message.

Since the decoding algorithms that we study use $\omega := \mathbf{wt}(\mathbf{e})$ as an input, we run the algorithms in the range $\omega \in [0, d]$ or $\omega \in [0, \lfloor \frac{d-1}{2} \rfloor]$, respectively. However, all our algorithms attain their maximum running time for their maximal weight $\omega = d$, respectively $\omega = \lfloor \frac{d-1}{2} \rfloor$. In the following we assume that we know ω .

Let us return to our syndrome decoding problem $\mathbf{H}\mathbf{e} = \mathbf{s}$. Naively, one can solve this equation by simply enumerating all weight- ω vectors $\mathbf{e} \in \mathbb{F}_2^n$ in time $\tilde{\mathcal{O}}\left(\binom{n}{\omega}\right)$.

Prange's Information Set Decoding: At the beginning of the 60s, Prange showed that the use of linear algebra provides a significant speedup. Notice that we can simply reorder the positions of the error vector \mathbf{e} by permuting the columns of \mathbf{H} . For some column permutation π let $\mathbf{Q} \in \mathbb{F}_2^{(n-k) \times (n-k)}$ denote the quadratic matrix at the right hand side of $\pi(\mathbf{H}) = (\cdot || \mathbf{Q})$. Assume that \mathbf{Q} has full rank, which happens with constant probability and define $\bar{\mathbf{s}} = \mathbf{Q}^{-1} \cdot \mathbf{s}$ and $\bar{\mathbf{H}} = \mathbf{Q}^{-1} \cdot \pi(\mathbf{H}) = (\cdot || \mathbf{I})$ for an $(n-k) \times (n-k)$ identity matrix \mathbf{I} . Let $\pi(\mathbf{e}) = \mathbf{e}_1 + (0^k || \mathbf{e}_q)$ with $\mathbf{e}_1 \in \mathbb{F}_2^k \times 0^{n-k}$ and $\mathbf{e}_q \in \mathbb{F}_2^{n-k}$ be the permuted error vector. Assume that $\mathbf{e}_1 = 0^n$. In this case, we call the first k error-free coordinates an *information set*. Having an information set, we can rewrite our syndrome equation as

$$\bar{\mathbf{H}}\pi(\mathbf{e}) = \bar{\mathbf{H}}\mathbf{e}_1 + \mathbf{e}_q = \bar{\mathbf{s}}, \text{ where } \mathbf{wt}(\mathbf{e}_1) = 0 \text{ and } \mathbf{wt}(\mathbf{e}_q) = \omega.$$

Since \mathbf{e}_1 is the zero vector, we can simplify as $\mathbf{e}_q = \bar{\mathbf{s}}$. Thus we only have to check whether $\bar{\mathbf{s}}$ has the correct weight $\mathbf{wt}(\bar{\mathbf{s}}) = \omega$.

Notice that all complexity in Prange's algorithm is moved to the initial permutation π of \mathbf{H} 's columns, whereas the remaining step has polynomial complexity. To improve upon the running time, it is reasonable to lower the restriction that the information set has no 1-entries in \mathbf{e}_1 , which was done in the work of Lee-Brickell [13]. Assume that the information set carries exactly p 1-positions. Then we enumerate over all $\binom{k}{p}$ possible $\mathbf{e}_1 \in \mathbb{F}_2^k \times 0^{n-k}$ with weight p . Therefore, we can test whether $\mathbf{wt}(\mathbf{e}_q) = \mathbf{wt}(\bar{\mathbf{s}} - \bar{\mathbf{H}}\mathbf{e}_1) = \omega - p$. On the downside, this trade-off between lowering the complexity for finding a good π and enumerating weight- p vectors does not pay off. Namely, asymptotically (in n) the trade-off achieves its optimum for $p = 0$.

On the positive side, we can improve on the simple enumeration of weight- p vectors by enumerating two lists of weight- $\frac{p}{2}$ vectors¹. This classical time-memory trade-off, usually called a Meet-in-the-middle approach, allows to reduce the enumeration time from $\binom{k}{p}$ to $\binom{k/2}{p/2}$ by increasing the memory to the same amount. Such a Meet-in-the-middle approach was introduced by Stern for Prange's Information Set decoding in 1989 [18]. In a nutshell, Stern's variant splits the first k columns of $\pi(\mathbf{e}) = \mathbf{e}_1 + \mathbf{e}_2 + (0^k || \mathbf{e}_q)$ with $\mathbf{e}_q \in \mathbb{F}_2^{n-k}$ in two parts $\mathbf{e}_1 \in \mathbb{F}_2^{k/2} \times 0^{k/2} \times 0^{n-k}$ and $\mathbf{e}_2 \in 0^{k/2} \times \mathbb{F}_2^{k/2} \times 0^{n-k}$. Additionally, we want a *good* permutation π to achieve

$$\bar{\mathbf{H}} \cdot (\mathbf{e}_1 + \mathbf{e}_2) + \mathbf{e}_q = \bar{\mathbf{s}} \text{ with } \mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p \text{ and } \mathbf{wt}(\mathbf{e}_q) = \omega - p. \quad (1)$$

Thus we have

$$\bar{\mathbf{H}}\mathbf{e}_1 = \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}} + \mathbf{e}_q. \quad (2)$$

Since $\mathbf{wt}(\mathbf{e}_q) = \omega - p$, for all but $\omega - p$ of the $n - k$ coordinates of the vectors we have

$$\bar{\mathbf{H}}\mathbf{e}_1 = \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}. \quad (3)$$

Remember that \mathbf{Q} was defined as the right hand part of $\pi(\mathbf{H})$. It can be shown that \mathbf{Q} is invertible with constant probability over the choice of \mathbf{H} and π . Thus inverting \mathbf{Q} can be ignored for the computation of the time complexity that suppresses polynomial factors.

Definition 1. A permutation π is good if \mathbf{Q} is invertible and if for $\pi(\mathbf{e})$ there exists a solution $(\mathbf{e}_1^*, \mathbf{e}_2^*)$ satisfying (1).

In Stern's algorithm, one computes for every candidate \mathbf{e}_1 the left-hand side of Eq. (3) and stores the result in a sorted list \mathbf{L} . Then, one computes for every \mathbf{e}_2 the right-hand side and looks whether the result is in \mathbf{L} . But recall that the above equation only holds for all but $\omega - p$ coordinates. Thus, one cannot simply match a candidate solution to one entry in \mathbf{L} .

The solution to this problem in Stern's algorithm is to introduce another parameter ℓ and to test whether there is an *exact* match on ℓ out of all $n - k$ coordinates. For those pairs $(\mathbf{e}_1, \mathbf{e}_2)$ whose result matches on ℓ coordinates, one checks whether they in total match on all but $\omega - p$ coordinates. However, this two-step approach for *approximately* matching similar vectors introduces another probability that enters the running time – namely that both vectors match on the chosen ℓ coordinates. Clearly, one would like to have some algorithm that directly addresses the approximate matching problem given by identity (2). Altogether, Stern's algorithm leads to the first asymptotical improvement since Prange's algorithm.

¹ Throughout the paper we ignore any rounding issues.

3 Our Decoding Algorithm

3.1 Application to Stern's Algorithm

Our main observation is that the matching in Stern's algorithm can be done in a smarter way by an algorithm for approximately matching vectors. We propose such an algorithm in Section 4. Our algorithm NEARESTNEIGHBOR works on two lists \mathbf{L}, \mathbf{R} with uniformly distributed and pairwise independent entries from \mathbb{F}_2^m , where one guarantees the existence of a pair from $\mathbf{L} \times \mathbf{R}$ that has small Hamming distance γm , $0 \leq \gamma \leq \frac{1}{2}$. On lists of equal size $|\mathbf{L}| = |\mathbf{R}|$ we achieve a running time that approaches $\tilde{\mathcal{O}}(|\mathbf{L}|^{\frac{1}{1-\gamma}})$. Notice that our running time is sub-quadratic for any $\gamma < \frac{1}{2}$. The smaller γ the better is our algorithm. In the case of $\gamma = 0$, we achieve linear running time (up to logarithmic factors) which coincides with the simple sort-and-match routine.

Our new matching algorithm immediately gives us an improved complexity for decoding random binary linear codes. First we proceed as in Stern's algorithm by enumerating over all weight- $\frac{p}{2}$ candidates for $\mathbf{e}_1 \in \mathbb{F}_2^{k/2} \times 0^{k/2} \times 0^{n-k}$. We compute the left-hand side $\bar{\mathbf{H}}\mathbf{e}_1$ of Eq. (3) and store the result in a list \mathbf{L} . For the right-hand side we proceed similar and store $\bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}$ in a list \mathbf{R} .

Notice that by construction each pair $(\mathbf{e}_1, \mathbf{e}_2)$ of enumerated error vectors yields an error vector $\mathbf{e}_1 + \mathbf{e}_2 + (0^k || \mathbf{e}_q)$ with $\mathbf{e}_q = \bar{\mathbf{H}}(\mathbf{e}_1 + \mathbf{e}_2) + \bar{\mathbf{s}}$ such that identity (2) holds. Moreover, by construction we have $\text{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p$. Thus all that remains is to find among all tuples $(\bar{\mathbf{H}}\mathbf{e}_1, \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}) \in \mathbf{L} \times \mathbf{R}$ one with small Hamming distance $\omega - p$.

Our complete algorithm DECODE is described in Algorithm 1.

Before we analyze correctness and running time of algorithm DECODE, we want to explain the idea of the subroutine NEARESTNEIGHBOR.

Basic Idea of NEARESTNEIGHBOR: Let m be the length of the vectors in \mathbf{L} and \mathbf{R} and define a constant λ such that $|\mathbf{L}| = |\mathbf{R}| = 2^{\lambda m}$. Assume the permutation π is *good* and hence $(\mathbf{e}_1^*, \mathbf{e}_2^*)$ exist such that $\text{wt}(\mathbf{e}_1^* + \mathbf{e}_2^*) = p$ and $\text{wt}(\bar{\mathbf{H}}\mathbf{e}_1^* + \bar{\mathbf{H}}\mathbf{e}_2^* + \bar{\mathbf{s}}) = \omega - p =: \gamma m$ for some $0 < \gamma < \frac{1}{2}$ holds. Let $\mathbf{u}^* := \bar{\mathbf{H}}\mathbf{e}_1^*$ and $\mathbf{v}^* := \bar{\mathbf{H}}\mathbf{e}_2^* + \bar{\mathbf{s}}$. Our algorithm NEARESTNEIGHBOR gets the input $(\mathbf{L}, \mathbf{R}, \gamma)$ and outputs a list \mathbf{C} , where $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{C}$ with overwhelming probability. Thus our algorithm solves the following problem.

Definition 2 (NN problem). Let $m \in \mathbb{N}$, $0 < \gamma < \frac{1}{2}$ and $0 < \lambda < 1$. In the (m, γ, λ) -Nearest Neighbor (NN) problem, we are given γ and two lists $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$ of equal size $2^{\lambda m}$ with uniform and pairwise independent vectors. If there exists a pair $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L} \times \mathbf{R}$ with Hamming distance $\Delta(\mathbf{u}^*, \mathbf{v}^*) = \gamma m$, we have to output a list \mathbf{C} that contains $(\mathbf{u}^*, \mathbf{v}^*)$.

Notice that in Definition 2 the lists \mathbf{L}, \mathbf{R} themselves do not have to be independent, e.g. $\mathbf{L} = \mathbf{R}$ is allowed. A naive algorithm solves the NN problem by simply computing the Hamming distance of each $(\mathbf{u}, \mathbf{v}) \in \mathbf{L} \times \mathbf{R}$ in quadratic time $\tilde{\mathcal{O}}((2^{\lambda m})^2)$. In the following we describe a sub-quadratic algorithm for the NN problem.

Algorithm 1. DECODE

```

1: procedure DECODE
2:   Input:  $n, k, \mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}, \mathbf{x} \in \mathbb{F}_2^n$ 
3:   Output:  $\mathbf{e} \in \mathbb{F}_2^n$  with  $\mathbf{He} = \mathbf{Hx}$  and  $\text{wt}(\mathbf{e}) \leq d$  (FDD),  $\text{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$  (HDD)
4:    $\mathbf{s} \leftarrow \mathbf{Hx}$                                       $\triangleright$  compute the syndrome
5:    $d \leftarrow H^{-1}(1 - \frac{k}{n}) \cdot n$        $\triangleright H$ : bin. entropy function, inverse  $H^{-1}$  maps to  $[0, \frac{1}{2}]$ 
6:   for  $\omega \leftarrow 0 \dots d$  do                   $\triangleright$  (FDD) or  $\omega \leftarrow 0 \dots \lfloor \frac{d-1}{2} \rfloor$  in the HDD case
7:     Choose  $0 < p < \omega$                        $\triangleright$  We find an optimal choice of  $p$  numerically
8:     repeat  $\text{poly}(n) \cdot \frac{1}{\mathbb{P}[\pi \text{ is good}]}$  many times
9:        $\pi \leftarrow$  random permutation on  $\mathbb{F}_2^n$ .
10:       $(\cdot || \mathbf{Q}) \leftarrow \pi(\mathbf{H})$  (permute columns) with  $\mathbf{Q} \leftarrow \mathbb{F}_2^{(n-k) \times (n-k)}$ 
11:      choose another permutation (goto line 9), if  $\mathbf{Q}$  is not invertible
12:       $\bar{\mathbf{H}} \leftarrow \mathbf{Q}^{-1}\pi(\mathbf{H})$  and  $\bar{\mathbf{s}} \leftarrow \mathbf{Q}^{-1}\mathbf{s}$ 
13:       $\mathbf{L} \leftarrow \bar{\mathbf{H}}\mathbf{e}_1$  for all  $\mathbf{e}_1 \in \mathbb{F}_2^{k/2} \times \mathbb{F}_2^{k/2} \times 0^{n-k}$  with  $\text{wt}(\mathbf{e}_1) = \frac{p}{2}$ 
14:       $\mathbf{R} \leftarrow \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}$  for all  $\mathbf{e}_2 \in \mathbb{F}_2^{k/2} \times \mathbb{F}_2^{k/2} \times 0^{n-k}$  with  $\text{wt}(\mathbf{e}_2) = \frac{p}{2}$ 
15:       $\mathbf{C} \leftarrow \text{NEARESTNEIGHBOR}(\mathbf{L}, \mathbf{R}, \frac{\omega-p}{n-k})$ 
16:      if  $(\mathbf{u}, \mathbf{v}) \in \mathbf{C} \cap (\mathbf{L} \times \mathbf{R})$  with Hamming distance  $\Delta(\mathbf{u}, \mathbf{v}) = \omega - p$  then
17:        find  $(\mathbf{e}_1, \mathbf{e}_2)$  s.t.  $\mathbf{u} = \bar{\mathbf{H}}\mathbf{e}_1$  and  $\mathbf{v} = \bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}$   $\triangleright$  binary search in  $\mathbf{L}, \mathbf{R}$ 
18:        return  $\pi^{-1}(\mathbf{e}_1 + \mathbf{e}_2 + (0^k || \mathbf{u} + \mathbf{v}))$ 
19:      end if
20:    until
21:  end for
22: end procedure

```

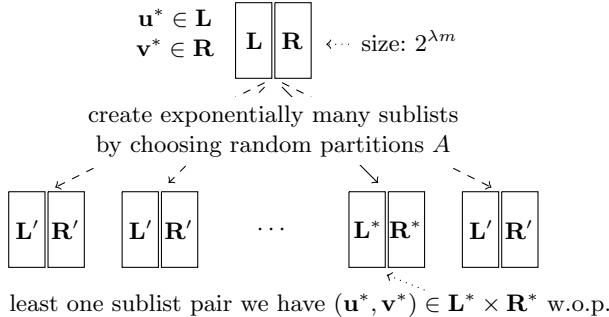
Given an initial list pair \mathbf{L}, \mathbf{R} , our main idea is to create exponentially many pairs of sublists \mathbf{L}', \mathbf{R}' . Each sublist is computed by first choosing a random partition $A \subset [m]$ of the columns of size $\frac{m}{2}$. We keep only those elements in \mathbf{L}', \mathbf{R}' that have a certain Hamming weight $h \cdot \frac{m}{2}$ on the columns defined by A , for some $0 < h < \frac{1}{2}$ that only depends on λ . The parameter h will be chosen s.t. each of the \mathbf{L}', \mathbf{R}' have expected polynomially (in m) many elements. We create as many sublists \mathbf{L}', \mathbf{R}' s.t. with overwhelming probability there exists a pair of sublists $\mathbf{L}^*, \mathbf{R}^*$ with $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L}^* \times \mathbf{R}^*$. For each sublist pair \mathbf{L}', \mathbf{R}' we check naively for a possible “good” vector by computing the Hamming distance $\Delta(\mathbf{u}', \mathbf{v}')$ for all $(\mathbf{u}', \mathbf{v}') \in \mathbf{L}' \times \mathbf{R}'$. Notice that this results only in a polynomial blow-up, because the list sizes are polynomial. We store all vectors (\mathbf{u}, \mathbf{v}) with the correct Hamming distance in the output list \mathbf{C} .

The idea of the algorithm is summarized in Fig. 4.

We will discuss the algorithm NEARESTNEIGHBOR in more detail in Section 4. The following theorem that we prove in Section 5 states its correctness and time complexity.

Theorem 1. *For any constant $\varepsilon > 0$ and any $\lambda < 1 - H(\frac{\gamma}{2})$, NEARESTNEIGHBOR solves the (m, γ, λ) -NN problem with overwhelming probability (over both the coins of the algorithm and the random choice of the input) in time*

$$\tilde{\mathcal{O}}\left(2^{(y+\varepsilon)m}\right) \quad \text{with} \quad y := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda) - \frac{\gamma}{2}}{1-\gamma}\right)\right).$$



For at least one sublist pair we have $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L}^* \times \mathbf{R}^*$ w.o.p.

Fig. 4. Main idea of our algorithm NEARESTNEIGHBOR

In Definition 2, we defined our list sizes $|\mathbf{L}| = |\mathbf{R}| = 2^{\lambda m}$ to be exponential in m , which is the cryptographically relevant scenario. A naive solution of the NN problem yields an exponent of $2\lambda m$, so we are interested in quotients $\frac{y}{\lambda} < 2$. In the following corollary we achieve a complexity of $\tilde{\mathcal{O}}(|\mathbf{L}|^{\frac{1}{1-\gamma}})$ in the case of polynomial list sizes $|\mathbf{L}| = |\mathbf{R}|$. This is the best case scenario for our algorithm, which is in the literature often referred to as the *unlimited amount of data* case. Notice that the quotient $\frac{y}{\lambda}$ is strictly increasing in λ until we reach the prerequisite bound $\lambda = 1 - H(\frac{\gamma}{2})$, beyond which our algorithm does no longer work. Finding a better dependency for the NN problem on λ would immediately result in further improvements for the decoding bounds from Theorems 2 and 3.

Corollary 1. *In the case of a list size $|\mathbf{L}| = |\mathbf{R}|$ that is polynomial in m , we obtain a complexity exponent $\lim_{\lambda \rightarrow 0} y/\lambda = \frac{1}{1-\gamma}$, i.e. our complexity is $\tilde{\mathcal{O}}(|\mathbf{L}|^{\frac{1}{1-\gamma}})$.*

Proof. Notice that we defined the inverse of the binary entropy function as $H^{-1}(\cdot)$ and that $H^{-1}(1) = \frac{1}{2}$. The derivative of the binary entropy function is $H'(x) = \log_2(\frac{1}{x} - 1)$ and the derivative of the inverse of the binary entropy function is $(H^{-1}(1 - \lambda))' = \frac{-1}{\log_2(\frac{1}{H^{-1}(1-\lambda)} - 1)}$. We obtain the result by the following calculation, using L'Hospital's rule twice.

$$\begin{aligned}
\lim_{\lambda \rightarrow 0} \frac{y}{\lambda} &= \lim_{\lambda \rightarrow 0} -(1 - \gamma) \log_2 \left(\frac{1 - \gamma}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} - 1 \right) \frac{1}{1 - \gamma} \frac{-1}{\log_2(\frac{1}{H^{-1}(1-\lambda)} - 1)} \\
&= \lim_{\lambda \rightarrow 0} \ln \left(\frac{1 - \gamma}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} - 1 \right) / \ln \left(\frac{1}{H^{-1}(1 - \lambda)} - 1 \right) \\
&= \lim_{\lambda \rightarrow 0} \frac{\left(\frac{1 - \gamma}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} - 1 \right)^{-1} \frac{(-1)(1-\gamma)}{(H^{-1}(1-\lambda)-\frac{\gamma}{2})^2} (H^{-1}(1 - \lambda))'}{\left(\frac{1}{H^{-1}(1 - \lambda)} - 1 \right)^{-1} \frac{(-1)}{(H^{-1}(1-\lambda))^2} (H^{-1}(1 - \lambda))'} \\
&= \lim_{\lambda \rightarrow 0} \frac{\frac{(-1)(1-\gamma)}{(H^{-1}(1-\lambda)-\frac{\gamma}{2})^2}}{\frac{(-1)}{(H^{-1}(1-\lambda))^2}} = \lim_{\lambda \rightarrow 0} (1 - \gamma) \left(\frac{H^{-1}(1 - \lambda)}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} \right)^2 = \frac{1}{1 - \gamma} \quad \square
\end{aligned}$$

In the following Theorem we show that DECODE is correct and prove its time complexity.

Theorem 2 (complexity and correctness). DECODE solves the decoding problem with overwhelming probability in time $\mathcal{O}(2^{0.114n})$ in the full and time $\mathcal{O}(2^{0.0550n})$ in the half distance decoding setting.

Proof. Let us define

$$\gamma := \frac{\omega - p}{n - k}, \quad r := H\left(\frac{\omega}{n}\right) - \frac{k}{n} \cdot H\left(\frac{p}{k}\right) - \left(1 - \frac{k}{n}\right) \cdot H(\gamma), \quad \mu := \frac{k}{2n} \cdot H\left(\frac{p}{k}\right)$$

and

$$y := (1 - \gamma) \left(1 - H\left(\frac{H^{-1}(1 - \frac{\mu n}{n-k}) - \frac{\gamma}{2}}{1 - \gamma}\right) \right).$$

We want to show that for any $\varepsilon > 0$ DECODE solves the decoding problem with overwhelming probability in time

$$\tilde{\mathcal{O}}\left(2^{rn} \left(2^{\mu n} + 2^{(y+\varepsilon)(n-k)}\right)\right). \quad (4)$$

In line 8 of DECODE, we repeat $\text{poly}(n) \cdot \frac{1}{\mathbb{P}[\pi \text{ is good}]}$ times. A permutation π is good, whenever $p/2$ ones are in the first $k/2$ columns, $p/2$ in the subsequent $k/2$ columns and $\omega - p$ ones in the last $n - k$ columns. Additionally, \mathbf{Q} (as defined in line 10) has to be invertible (which happens with constant probability). Therefore, the number of repetitions until we find a good π is

$$\text{poly}(n) \cdot \frac{\binom{n}{\omega}}{\binom{k/2}{p/2}^2 \cdot \binom{n-k}{\gamma(n-k)}} = \tilde{\mathcal{O}}\left(2^{n \cdot H(\omega/n) - k \cdot H(p/k) - (n-k) \cdot H(\gamma)}\right) = \tilde{\mathcal{O}}(2^{rn}).$$

We fix a repetition that leads to a good permutation π . In this repetition, we therefore have $\mathbf{e}_1^*, \mathbf{e}_2^*$ with $\mathbf{wt}(\mathbf{e}_1^* + \mathbf{e}_2^*) = p$ and $\Delta(\mathbf{u}^*, \mathbf{v}^*) = \omega - p =: \gamma(n - k)$ with $\mathbf{u}^* := \bar{\mathbf{H}}\mathbf{e}_1^*$ and $\mathbf{v}^* := \bar{\mathbf{H}}\mathbf{e}_2^* + \bar{\mathbf{s}}$. In lines 13 and 14, the algorithm creates two lists of uniform and pairwise independent vectors of size $n - k$ s.t. by construction $\mathbf{u}^* \in \mathbf{L}$ and $\mathbf{v}^* \in \mathbf{R}$ and

$$|\mathbf{L}| = |\mathbf{R}| = \binom{k/2}{p/2} = \tilde{\mathcal{O}}\left(2^{k/2 \cdot H(p/k)}\right) = \tilde{\mathcal{O}}(2^{\mu n}).$$

Notice that $\mu n = \lambda(n - k)$ controls the list size. By a suitably small choice of p one can always satisfy the prerequisite $\lambda < 1 - H(\frac{\gamma}{2})$ of Theorem 1. Thus we obtain an instance $(\mathbf{L}, \mathbf{R}, \gamma)$ of the $(n - k, \gamma, \mu \frac{n}{n-k})$ -NN problem, for which Theorem 1 guarantees to output a list \mathbf{C} that contains the solution $(\mathbf{u}^*, \mathbf{v}^*)$ with overwhelming probability. Notice that any vector $(\mathbf{u}, \mathbf{v}) \in \mathbf{C} \cap (\mathbf{L} \times \mathbf{R})$ with a Hamming distance of $\omega - p$ solves our problem, because the corresponding \mathbf{e}_1 with $\bar{\mathbf{H}}\mathbf{e}_1 = \mathbf{u}$ and \mathbf{e}_2 with $\bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}} = \mathbf{v}$ have the property $\mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p$. This in turn leads to $\mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2 + (0^k || (\mathbf{u} + \mathbf{v})) = \omega$. In line 17, the $(\mathbf{e}_1, \mathbf{e}_2)$ can be

found by a binary search in slightly modified lists \mathbf{L}, \mathbf{R} (that also store \mathbf{e}_1 and \mathbf{e}_2). Thus, with overwhelming probability, the algorithm outputs a solution to the decoding problem in line 18.

Also by Theorem 1, an application of algorithm NEARESTNEIGHBOR has time complexity $\tilde{\mathcal{O}}(2^{(y+\varepsilon)(n-k)})$ for any $\varepsilon > 0$. Notice that this complexity is independent of whether we have a good permutation π or not. This complexity has to be added to the creation time of the input lists \mathbf{L}, \mathbf{R} . Recall that the loop has to be repeated $\tilde{\mathcal{O}}(2^{rn})$ times, leading to the runtime of Eq. (4).

Numerical optimization in the half distance decoding case yields time complexity $\mathcal{O}(2^{0.0550n})$ in the worst case $k/n \approx 0.466$ with $p/n \approx 0.00383$. In the full distance decoding case we get a runtime of $\mathcal{O}(2^{0.114n})$ in the worst case $k/n \approx 0.447$ with $p/n \approx 0.01286$. \square

3.2 Application to the BJMM Algorithm

It is possible to apply our idea to the decoding algorithm of Becker, Joux, May and Meurer (BJMM) [4]. We already explained that BJMM is a variant of Stern's algorithm and thus a Meet-in-the-Middle algorithm that constructs two lists $\tilde{\mathbf{L}}, \tilde{\mathbf{R}}$. The major difference is that $\tilde{\mathbf{L}}, \tilde{\mathbf{R}}$ are not directly enumerated as the lists \mathbf{L}, \mathbf{R} in Stern's algorithm. Instead, $\tilde{\mathbf{L}}, \tilde{\mathbf{R}}$ are constructed in a more involved tree-based manner. This has the benefit that the list length is significantly smaller than in Stern's construction, which in turn leads to an improved running time. This similarity however enables us to directly apply our technique to the BJMM algorithm. Namely, we have to simply replace in DECODE the construction of \mathbf{L}, \mathbf{R} by the BJMM-construction of $\tilde{\mathbf{L}}, \tilde{\mathbf{R}}$, on which we apply our NEARESTNEIGHBOR-algorithm.

Notice that as opposed to Section 3.1 not all possible vector pairs in \mathbf{C} with the correct Hamming distance solve the decoding problem. The issue is that the corresponding $\mathbf{e}_1, \mathbf{e}_2$ do not necessarily have a Hamming distance of p . Thus, additionally to $\Delta(\mathbf{u}, \mathbf{v}) = \omega - p$, we have to verify that also $\Delta(\mathbf{e}_1, \mathbf{e}_2) = p$ holds.

Algorithm DECODEBJMM describes the application of our algorithm in the BJMM framework. Notice that up to line 22 the algorithm is identical to the one in [4]. The only difference is the final step (from line 23). Instead of using the exact matching of Stern, we use our NEARESTNEIGHBOR algorithm that searches for two vectors that are close (i.e. have Hamming distance $\omega - p$) on the remaining $n - k - \ell$ coordinates.

Algorithm DECODEBJMM uses a subroutine BASELISTS. As described in [4], BASELISTS chooses a random partition of the first $k + \ell$ columns into two sets $P_1, P_2 \subseteq [k + \ell]$ of equal size. Then a list \mathbf{B}_1 is created that contains all vectors $\mathbf{b}_1 \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$ with $\text{wt}(\mathbf{b}_1) = \frac{p}{8} + \frac{\varepsilon_1}{4} + \frac{\varepsilon_2}{2}$ that are zero on the coordinates from P_2 . Analogously, a list \mathbf{B}_2 is build, that contains all vectors $\mathbf{b}_2 \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$ of the same Hamming weight as above, but with zeros on the coordinates from P_1 . Thus, with inverse polynomial probability, a fixed vector of size $k + \ell$ with Hamming weight $\frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2$ can be represented as a sum of an element in \mathbf{B}_1 and an element in \mathbf{B}_2 . Repeating this choice a polynomial number of times guarantees the representation with overwhelming probability.

Algorithm 2. DECODEBJMM

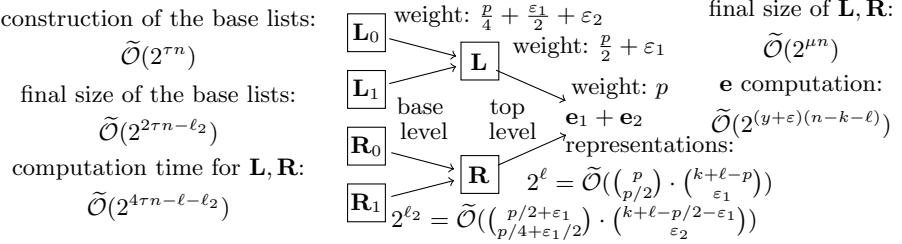
```

1: procedure DECODEBJMM
2:   Input:  $n, k, \mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$ ,  $\mathbf{x} \in \mathbb{F}_2^n$ 
3:   Output:  $\mathbf{e} \in \mathbb{F}_2^n$  with  $\mathbf{He} = \mathbf{Hx}$  and  $\text{wt}(\mathbf{e}) \leq d$  (FDD),  $\text{wt}(\mathbf{e}) \leq \lfloor \frac{d-1}{2} \rfloor$  (HDD)
4:    $\mathbf{s} \leftarrow \mathbf{Hx}$ 
5:    $d \leftarrow H^{-1}(1 - \frac{k}{n}) \cdot n$      $\triangleright H$ : bin. entropy function, inverse  $H^{-1}$  maps to  $[0, \frac{1}{2}]$ .
6:   for  $\omega \leftarrow 0 \dots d$  do           $\triangleright$  (FDD) or  $\omega \leftarrow 0 \dots \lfloor \frac{d-1}{2} \rfloor$  in the HDD case
7:     Choose  $0 < p, \varepsilon_1, \varepsilon_2 < \omega$ ,  $0 < \ell_2 < \ell < n - k$            $\triangleright$  optimize numerically
8:     repeat  $\text{poly}(n) \cdot \frac{1}{\mathbb{P}[\pi \text{ is good}]}$  many times
9:        $\pi \leftarrow$  random permutation on  $\mathbb{F}_2^n$ .
10:       $(\cdot || \mathbf{Q}) \leftarrow \pi(\mathbf{H})$  (permute columns) with  $\mathbf{Q} \leftarrow \mathbb{F}_2^{(n-k) \times (n-k)}$ 
11:      choose another permutation (goto line 9), if  $\mathbf{Q}$  is not invertible
12:       $\bar{\mathbf{H}} \leftarrow \mathbf{Q}^{-1}\pi(\mathbf{H})$  and  $\bar{\mathbf{s}} \leftarrow \mathbf{Q}^{-1}\mathbf{s}$ 
13:       $\mathbf{t}_L \in_R \mathbb{F}_2^\ell, \mathbf{t}_{L_0}, \mathbf{t}_{L_1}, \mathbf{t}_{R_0} \in_R \mathbb{F}_2^{\ell_2}$            $\triangleright$  choose uniformly at random
14:       $\mathbf{t}_R = [\bar{\mathbf{s}}]_\ell - \mathbf{t}_L$            $\triangleright [\cdot]_c$  restricts to first  $c$  columns
15:       $\mathbf{t}_{R_1} = [\bar{\mathbf{s}}]_{\ell_2} - \mathbf{t}_{L_0} - \mathbf{t}_{L_1} - \mathbf{t}_{R_0}$            $\triangleright [\cdot]_c$  restricts to last  $c$  columns
16:       $\mathbf{L}_0 \leftarrow \text{BASELISTS}(\bar{\mathbf{H}}, p, \varepsilon_1, \varepsilon_2, \mathbf{t}_{L_0})$            $\triangleright$  list of  $\mathbf{b} \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$ 
17:       $\mathbf{L}_1 \leftarrow \text{BASELISTS}(\bar{\mathbf{H}}, p, \varepsilon_1, \varepsilon_2, \mathbf{t}_{L_1})$            $\triangleright$  with  $\text{wt}(\mathbf{b}) = \frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2$ 
18:       $\mathbf{R}_0 \leftarrow \text{BASELISTS}(\bar{\mathbf{H}}, p, \varepsilon_1, \varepsilon_2, \mathbf{t}_{R_0})$            $\triangleright$  s.t.  $[\bar{\mathbf{H}}\mathbf{b}]_{\ell_2} = \mathbf{t}_{L_0}$ 
19:       $\mathbf{R}_1 \leftarrow \text{BASELISTS}(\bar{\mathbf{H}}, p, \varepsilon_1, \varepsilon_2, \mathbf{t}_{R_1})$ 
20:       $\mathbf{L} \leftarrow [\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y})]^{n-k-\ell}$  for all  $\mathbf{x} \in \mathbf{L}_0, \mathbf{y} \in \mathbf{L}_1$  with  $[\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y})]_\ell = \mathbf{t}_L$ 
21:       $\mathbf{R} \leftarrow [\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y}) + \bar{\mathbf{s}}]^{n-k-\ell}$  for all  $\mathbf{x} \in \mathbf{R}_0, \mathbf{y} \in \mathbf{R}_1$  with  $[\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y})]_\ell = \mathbf{t}_R$ 
22:      (In lines 20, 21: only keep elements with  $\text{wt}(\mathbf{x} + \mathbf{y}) = \frac{p}{2} + \varepsilon_1$ )
23:       $\mathbf{C} \leftarrow \text{NEARESTNEIGHBOR}(\mathbf{L}, \mathbf{R}, \frac{\omega-p}{n-k-\ell})$ 
24:      for all  $(\mathbf{u}, \mathbf{v}) \in \mathbf{C} \cap (\mathbf{L} \times \mathbf{R})$  with distance  $\Delta(\mathbf{u}, \mathbf{v}) = \omega - p$  do
25:        find  $(\mathbf{e}_1, \mathbf{e}_2)$  s.t.  $\mathbf{u} = [\bar{\mathbf{H}}\mathbf{e}_1]^{n-k-\ell}$  and  $\mathbf{v} = [\bar{\mathbf{H}}\mathbf{e}_2 + \bar{\mathbf{s}}]^{n-k-\ell}$ 
26:        if  $\text{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p$  then
27:          return  $\pi^{-1}(\mathbf{e}_1 + \mathbf{e}_2 + (0^{k+\ell} || \mathbf{u} + \mathbf{v}))$ 
28:        end if
29:      end for
30:    until
31:  end for
32: end procedure

```

BASELISTS continues by computing the corresponding values $\bar{\mathbf{H}}\mathbf{b}_1$ for each element $\mathbf{b}_1 \in \mathbf{B}_1$, stores these elements and sorts the list by these values. Eventually an output list of vectors in $\mathbf{b} \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$ with weight $\frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2$ is computed by a standard meet-in-the-middle technique s.t. $\bar{\mathbf{H}}\mathbf{b}$ equals the input target value \mathbf{t} on the first ℓ_2 coordinates.

The same technique is used in lines 20 and 21. In the computation of \mathbf{L} , for each pair of vectors $(\mathbf{x}, \mathbf{y}) \in \mathbf{L}_0 \times \mathbf{L}_1$ a list of sums $\mathbf{x} + \mathbf{y}$ is obtained such that $\bar{\mathbf{H}}(\mathbf{x} + \mathbf{y})$ matches a uniformly chosen target value \mathbf{t}_L on the first ℓ coordinates. After this step we also restrict to only those elements that have a certain Hamming weight of $\frac{p}{2} + \varepsilon_1$, since by [4] the target solution splits in two vectors of this particular weight. The computation tree is illustrated in Figure 5.

**Fig. 5.** Computation tree of DECODEBJMM

Theorem 3. DECODEBJMM solves the decoding problem with overwhelming probability in time $\mathcal{O}(2^{0.097n})$ in the full distance decoding setting and time $\mathcal{O}(2^{0.0473n})$ in the half distance decoding setting.

Proof. Let us define

$$\gamma := \frac{\omega - p}{n - k - \ell}, \quad r := H\left(\frac{\omega}{n}\right) - \frac{k + \ell}{n} \cdot H\left(\frac{p}{k + \ell}\right) - \left(1 - \frac{k + \ell}{n}\right) \cdot H(\gamma),$$

$$\tau := \frac{k + \ell}{2n} \cdot H\left(\frac{\frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2}{k + \ell}\right), \quad \ell \stackrel{!}{=} p + (k + \ell - p) \cdot H\left(\frac{\varepsilon_1}{k + \ell - p}\right)$$

$$\mu := \frac{k + \ell}{n} \cdot H\left(\frac{\frac{p}{2} + \varepsilon_1}{k + \ell}\right) - \frac{\ell}{n}, \quad \ell_2 := \frac{p}{2} + \varepsilon_1 + (k + \ell - \frac{p}{2} - \varepsilon_1) \cdot H\left(\frac{\varepsilon_2}{k + \ell - \frac{p}{2} - \varepsilon_1}\right)$$

and

$$y := (1 - \gamma) \left(1 - H\left(\frac{H^{-1}(1 - \frac{\mu n}{n - k - \ell}) - \frac{\gamma}{2}}{1 - \gamma}\right) \right).$$

We want to show that for any $\varepsilon > 0$ the decoding problem can be solved with overwhelming probability in time

$$\tilde{\mathcal{O}}\left(2^{rn} \left(2^{\tau n} + 2^{2\tau n - \ell_2} + 2^{4\tau n - \ell - \ell_2} + 2^{\mu n} + 2^{(y+\varepsilon)(n-k-\ell)}\right)\right).$$

The correctness and time complexity of the first part of the algorithm (i.e. the computation of \mathbf{L} and \mathbf{R} up to line 22) was already shown in [4]. Let us summarize the time complexity for this computation. First of all we have a loop that guarantees a *good* distribution with overwhelming probability. In our case, a good splitting would be p ones on the first $k + \ell$ coordinates and $\omega - p$ ones on the remaining $n - k - \ell$ coordinates. Thus the necessary number of repetitions is $\tilde{\mathcal{O}}((\frac{n}{\omega}) / [(\frac{k+\ell}{p}) \cdot (\frac{n-k-\ell}{\omega-p})]) = \tilde{\mathcal{O}}(2^{rn})$.

The algorithm of BJMM [4] makes use of the so-called *representation technique* introduced by Joux and Howgrave-Graham [11]. The main idea is to blow up the search space such that each error vector can be represented as a sum of two vectors in many different ways. In the algorithm, all but one of these

representations are filtered out using some restriction. Inside the loop, we therefore first need to make sure that 2^ℓ is the number of *representations* on the top level, because we restrict to ℓ binary coordinates. On the top level we split the $\mathbb{F}_2^{k+\ell}$ vector with p ones in a sum of two vectors in $\mathbb{F}_2^{k+\ell}$ with $\frac{p}{2} + \varepsilon_1$ ones each. Thus there are $\binom{p}{p/2}$ ways to represent the ones (as $1+0$ or $0+1$) and $\binom{k+\ell-p}{\varepsilon_1}$ ways to represent the zeros (as $0+0$ or $1+1$). Hence we need to choose ℓ such that $2^\ell = \tilde{\Theta}(\binom{p}{p/2} \cdot \binom{k+\ell-p}{\varepsilon_1})$. On the bottom level, vectors with $\frac{p}{2} + \varepsilon_1$ ones are represented as sums of two vectors with $\frac{p}{4} + \frac{\varepsilon_1}{2} + \varepsilon_2$ ones each. In this case there are $\binom{p/2+\varepsilon_1}{p/4+\varepsilon_1/2}$ ways to represent the ones and $\binom{k+\ell-p/2-\varepsilon_1}{\varepsilon_2}$ ways to represent the zeros. Thus we choose $2^{\ell_2} = \tilde{\Theta}(\binom{p/2+\varepsilon_1}{p/4+\varepsilon_1/2} \cdot \binom{k+\ell-p/2-\varepsilon_1}{\varepsilon_2})$.

The computation starts by creating the four base lists. In the first step two lists with vectors of size $\frac{k+\ell}{2}$ and $\frac{p}{8} + \frac{\varepsilon_1}{4} + \frac{\varepsilon_2}{2}$ ones are created, which takes time $\tilde{\mathcal{O}}(\binom{\frac{k+\ell}{2}}{\frac{p}{8} + \frac{\varepsilon_1}{4} + \frac{\varepsilon_2}{2}}) = \tilde{\mathcal{O}}(2^{\tau n})$. These two lists are merged, considering each pair of one vector of the first list and one vector of the second list such that the first ℓ_2 coordinates of the sum are a fixed value (i.e. restricting to one special *representation*). The number of elements in the base lists is therefore $\tilde{\mathcal{O}}(2^{2\tau n}/2^{\ell_2})$.

In lines 20 and 21 the top level lists **L** and **R** are computed from the base lists. In this step, the vectors are restricted to additional $\ell - \ell_2$ coordinates, resulting in a total restriction of 2^ℓ . Therefore, the time complexity is $\tilde{\mathcal{O}}((2^{2\tau n}/2^{\ell_2})^2/(2^{\ell-\ell_2})) = \tilde{\mathcal{O}}(2^{4\tau n-\ell-\ell_2})$.

In line 22 the algorithm restricts the lists **L** and **R** to those vectors with $\frac{p}{2} + \varepsilon_1$ ones. Due to the fact that the vectors are restricted to fixed ℓ coordinates, the number of elements can be upper bounded by $\tilde{\mathcal{O}}(\binom{k+\ell}{p/2+\varepsilon_1}/2^\ell) = \tilde{\mathcal{O}}(2^{\mu n})$.

In the final step we have two lists of uniform (because the elements are a linear combination of the columns of **H**) and pairwise independent (because each element is computed by a linear combination of pairwise different columns) vectors in $\mathbb{F}_2^{n-k-\ell}$.

From the analysis in [4] we know that there are $\mathbf{e}_1^*, \mathbf{e}_2^* \in \mathbb{F}_2^{k+\ell} \times 0^{n-k-\ell}$ with $\mathbf{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p$ such that $\mathbf{u}^* = [\bar{\mathbf{H}}\mathbf{e}_1^*]^{n-k-\ell} \in \mathbf{L}$ and $\mathbf{v}^* = [\bar{\mathbf{H}}\mathbf{e}_2^* + \bar{\mathbf{s}}]^{n-k-\ell} \in \mathbf{R}$, where $\mathbf{wt}(\mathbf{u}^* + \mathbf{v}^*) = \omega - p$. Therefore, by Theorem 1, NEARESTNEIGHBOR outputs a list **C** that contains $(\mathbf{u}^*, \mathbf{v}^*)$. The $(\mathbf{e}_1^*, \mathbf{e}_2^*)$ can be found in line 25 by binary searching in slightly modified **L**, **R** (that also contain $\mathbf{x} + \mathbf{y}$).

Thus $\pi^{-1}(\mathbf{e}_1^* + \mathbf{e}_2^* + (0^{k+\ell}||\mathbf{u}^* + \mathbf{v}^*))$ (with weight ω) is a correct solution to the problem, because

$$\begin{aligned} \bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^* + (0^{k+\ell}||\mathbf{u}^* + \mathbf{v}^*)) &= \bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + (0^\ell||\mathbf{u}^* + \mathbf{v}^*) \\ &= \bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + (0^\ell||[\bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + \bar{\mathbf{s}}]^{n-k-\ell}) \\ &= \bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + (\bar{\mathbf{H}} \cdot (\mathbf{e}_1^* + \mathbf{e}_2^*) + \bar{\mathbf{s}}) = \bar{\mathbf{s}}. \end{aligned}$$

Notice that $[\bar{\mathbf{H}}(\mathbf{e}_1^* + \mathbf{e}_2^*) + \bar{\mathbf{s}}]_\ell = 0^\ell$ holds by construction. By Theorem 1 the time complexity of NEARESTNEIGHBOR is $2^{(y+\varepsilon)(n-k-\ell)}$.

In the half distance decoding case, for the worst case $k/n \approx 0.45$ we get a time complexity of $\mathcal{O}(2^{0.0473n})$ with $p/n \approx 0.01667$, $\varepsilon_1/n \approx 0.00577$ and $\varepsilon_2/n \approx$

0.00124. In the full distance decoding setting, we have a runtime of $\mathcal{O}(2^{0.097n})$ with $k/n \approx 0.42, p/n \approx 0.06284, \varepsilon_1/n \approx 0.02001$ and $\varepsilon_2/n \approx 0.00391$. \square

4 Solving the Nearest Neighbor Problem

In this section we will describe NEARESTNEIGHBOR, an algorithm that solves the Nearest Neighbor problem from Definition 2. We will prove correctness and time complexity of our algorithm in the subsequent section.

As already outlined in the previous section, given the input lists \mathbf{L} and \mathbf{R} with $|\mathbf{L}| = |\mathbf{R}| = 2^{\lambda m}$, our idea is to create exponentially many sublists \mathbf{L}', \mathbf{R}' that are of expected polynomial size. The sublists are chosen such that with overwhelming probability our unknown solution $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L} \times \mathbf{R}$ with $\Delta(\mathbf{u}^*, \mathbf{v}^*) = \gamma m$ is contained in at least one of these sublists. The sublists \mathbf{L}' (resp. \mathbf{R}') are defined as all elements of \mathbf{L} (resp. \mathbf{R}) that have a Hamming weight of $h \frac{m}{2}$ on the columns defined by a random partition $A \subset [m]$ of size $\frac{m}{2}$. In Lemma 3, we will prove that $h := H^{-1}(1 - \lambda)$ with $0 \leq h \leq \frac{1}{2}$ is a suitable choice, because it leads to sublists of expected polynomial size.

We will prove in Lemma 2 that the required number of sublists such that $(\mathbf{u}^*, \mathbf{v}^*)$ is contained in one of these sublists is $\tilde{\mathcal{O}}(2^{ym})$ with

$$y := (1 - \gamma) \left(1 - H \left(\frac{H^{-1}(1 - \lambda) - \frac{\gamma}{2}}{1 - \gamma} \right) \right). \quad (5)$$

We will make use of the fact that $y > \lambda$ for any constant $0 < \lambda < 1, 0 < \gamma < \frac{1}{2}$, which can be verified numerically.

There is still one problem to solve, because we never discussed how to compute the \mathbf{L}', \mathbf{R}' given \mathbf{L}, \mathbf{R} . A naive way to do so would be to traverse the original lists linearly and to check the weight condition. Unfortunately, this would have to be done for each sampled A , which would result in an overall complexity of $\tilde{\mathcal{O}}(2^{ym} \cdot 2^{\lambda m})$.

Instead, as illustrated in Fig. 6, we do not proceed with the whole m coordinates at once, but first start with a strip $\{1, \dots, \alpha_1 m\}$ of the left hand side columns and filter only on that strip. The resulting list pairs $\mathbf{L}^1, \mathbf{R}^1$ are still of exponential, but smaller, size. In the second step, we proceed on the subsequent $\alpha_2 m$ columns $\{\alpha_1 m + 1, \dots, (\alpha_1 + \alpha_2)m\}$ to generate sublists $\mathbf{L}^2, \mathbf{R}^2$ that contain all elements that in addition have a small Hamming weight on the second strip. The advantage of this technique is that we are able to use the smaller lists $\mathbf{L}^1, \mathbf{R}^1$ to construct $\mathbf{L}^2, \mathbf{R}^2$, again by traversing these lists, instead of using \mathbf{L}, \mathbf{R} for all the m columns.

As illustrated in Fig. 7, we grow a search tree of *constant* depth t , where the leaves are pairs of lists $\mathbf{L}^t, \mathbf{R}^t$, which were filtered on $(\alpha_1 + \dots + \alpha_t)m$ coordinates. We choose $\alpha_1 + \dots + \alpha_t = 1$ to cover all coordinates. The choice of the α_j will be given in Theorem 1 and is basically done in a way to balance the costs in each level of the search tree, which allows us to solve the problem in

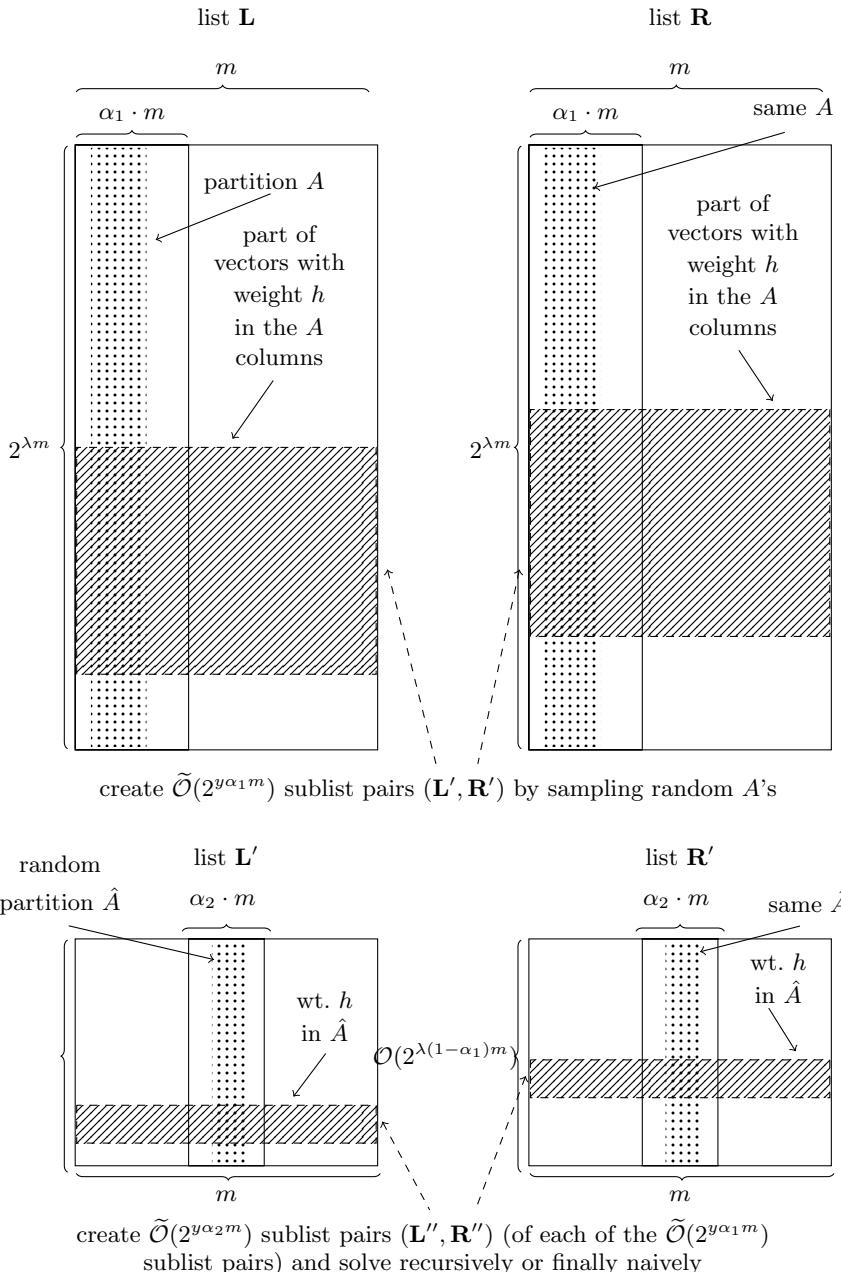


Fig. 6. Step-by-step computation of NEARESTNEIGHBOR

time $\tilde{\mathcal{O}}(2^{(y+\varepsilon)m})$ for any constant $\varepsilon > 0$. Notice that we never actually compute the Hamming distance between elements from the list pairs, except for the very last step, where we obtain list pairs $\mathbf{L}^t, \mathbf{R}^t$ of small size $\tilde{\mathcal{O}}(2^{\frac{\varepsilon}{2}m})$, and compute the Hamming distance of all pairs by a naive quadratic algorithm, which has time complexity $\tilde{\mathcal{O}}(2^{\varepsilon m})$. Because we proceed analogously for any of the $\tilde{\mathcal{O}}(2^{ym})$ sublists, we obtain an overall time complexity of $\tilde{\mathcal{O}}(2^{(y+\varepsilon)m})$.

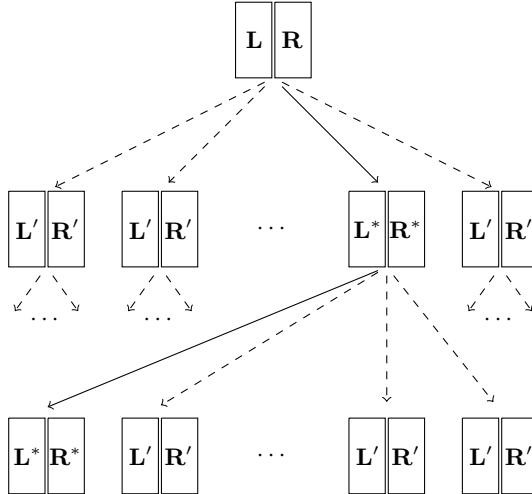


Fig. 7. Example: computation tree of depth $t = 2$ with one good path (\rightarrow)

In total, our algorithm heavily relies on the observation that the property of the pair $(\mathbf{u}^*, \mathbf{v}^*)$ with small Hamming distance $\Delta(\mathbf{u}^*, \mathbf{v}^*) = \gamma m$ also holds *locally* on each of the t strips. In case that the differing coordinates of $(\mathbf{u}^*, \mathbf{v}^*)$ would cluster in any of the strips $\alpha_j m$, we would by mistake sort out the pair. However, this issue can be easily resolved by rerandomizing the position of the coordinates in both input lists \mathbf{L} and \mathbf{R} . Denote $\mathbf{z}^* := \mathbf{u}^* + \mathbf{v}^* \in \mathbb{F}_2^m$ and the splitting $\mathbf{z}^* = (\mathbf{z}_1^*, \dots, \mathbf{z}_t^*) \in \mathbb{F}_2^{\alpha_1 m} \times \mathbb{F}_2^{\alpha_2 m} \times \dots \times \mathbb{F}_2^{\alpha_t m}$ according to the t strips. We will prove in Lemma 1 that after randomly permuting the m columns a polynomial in m number of times, there will be one permutation such that $\text{wt}(\mathbf{z}_j) = \gamma \alpha_j m$ for all $1 \leq j \leq t$.

Furthermore, we want to enforce that $\text{wt}(\mathbf{u}^*) = \text{wt}(\mathbf{v}^*) = \frac{1}{2}m$, which also has to hold on each of the t strips. Define $\mathbf{u}^* = (\mathbf{u}_1^*, \dots, \mathbf{u}_t^*)$ and $\mathbf{v}^* = (\mathbf{v}_1^*, \dots, \mathbf{v}_t^*)$ as above. We therefore want to make sure that $\text{wt}(\mathbf{u}_j^*) = \text{wt}(\mathbf{v}_j^*) = \frac{1}{2}\alpha_j m$ for all $1 \leq j \leq t$. Notice that this can also be achieved by rerandomization. Concretely, we pick a uniformly random vector $\mathbf{r} \in \mathbb{F}_2^m$ and add this vector to all elements of both input lists \mathbf{L}, \mathbf{R} . Notice that the Hamming weight of our solution pair $(\mathbf{u}^*, \mathbf{v}^*)$ isn't changed by this operation. We will also show in Lemma 1 that

after applying this process a polynomial (in m) number of times, the vectors \mathbf{u}^* and \mathbf{v}^* have the desired Hamming weight in at least one step.

Algorithm 3. NEARESTNEIGHBOR

```

1: procedure NEARESTNEIGHBOR( $\mathbf{L}, \mathbf{R}, \gamma$ ) ▷  $\mathbf{L}, \mathbf{R} \subset \mathbb{F}_2^m$ ,  $0 < \gamma < \frac{1}{2}$ 
2:   compute vectors length  $m$  and size  $\lambda$  from  $\mathbf{L}, \mathbf{R}$ 
3:    $y := (1 - \gamma) \left( 1 - H \left( \frac{H^{-1}(1-\lambda)-\frac{\gamma}{2}}{1-\gamma} \right) \right)$  ▷ as defined in Theorem 1
4:   choose a constant  $\varepsilon > 0$  ▷ could also be an input
5:    $t := \lceil \frac{\log(y-\lambda+\frac{\varepsilon}{2}) - \log(\frac{\varepsilon}{2})}{\log(y)-\log(\lambda)} \rceil$  ▷ as defined in the proof of Theorem 1
6:    $\alpha_1 := \frac{y-\lambda+\frac{\varepsilon}{2}}{y}$  ▷ as defined in the proof of Theorem 1
7:   for  $2 \leq j \leq t$  do
8:      $\alpha_i := \frac{y}{\lambda} \cdot \alpha_{i-1}$  ▷ as defined in the proof of Theorem 1
9:   end for
10:  for  $\text{poly}(m)$  uniformly random permutations  $\pi$  of  $[m]$  do
11:    for  $\text{poly}(m)$  unif. rand.  $\mathbf{r} \in \mathbb{F}_2^{\alpha_1 m} \times \dots \times \mathbb{F}_2^{\alpha_t m}$  (wt.  $\alpha_j \frac{m}{2}$  on each strip) do
12:       $\bar{\mathbf{L}} \leftarrow \pi(\mathbf{L}) + \mathbf{r}$ 
13:       $\bar{\mathbf{R}} \leftarrow \pi(\mathbf{R}) + \mathbf{r}$  ▷ permute columns and add  $\mathbf{r}$  to all elements
14:      Remove all vectors from  $\bar{\mathbf{L}}, \bar{\mathbf{R}}$  that are not of weight  $\alpha_j \frac{m}{2}$  on each strip
15:      return NEARESTNEIGHBORREC( $\bar{\mathbf{L}}, \bar{\mathbf{R}}, m, t, \gamma, \lambda, \alpha_1, \dots, \alpha_t, y, \varepsilon, 1$ )
16:    end for
17:  end for
18: end procedure

```

Our algorithm NEARESTNEIGHBOR starts by computing the length of the vectors m and a λ such that $|\mathbf{L}| = |\mathbf{R}| = 2^{\lambda m}$ from \mathbf{L} and \mathbf{R} . This list size is used to compute the repetition parameter y . The algorithm chooses a constant $\varepsilon > 0$ that is part of the asymptotic time complexity. The parameter ε also determines the number of strips t and the relative sizes of the strips $\alpha_1, \dots, \alpha_t$. Eventually, the two input lists are rerandomized by permuting the columns and adding a random vector. Another recursive algorithm NEARESTNEIGHBORREC is then called with the rerandomized lists as input.

In the algorithm NEARESTNEIGHBORREC we sample random partitions A of the columns, until it is guaranteed with overwhelming probability that the solution is in at least one of the created sublists. The sublists are created by naively traversing the input lists for all vectors with a Hamming weight of $H^{-1}(1-\lambda)\frac{\alpha_j m}{2}$ on the columns defined by the random partition. We only continue, if \mathbf{L}' and \mathbf{R}' don't grow too large, as defined in Lemma 3. In line 10, we apply the algorithm recursively on the subsequent $\alpha_2 m$ columns, and so on. Eventually, we compare the final list pairs naively.

5 Analysis of Our Algorithm

In this section, we first show that NEARESTNEIGHBOR achieves a good distribution in at least one of the polynomially many repetitions. We define a *good*

Algorithm 4. NEARESTNEIGHBORREC

```

1: procedure NEARESTNEIGHBORREC( $\mathbf{L}, \mathbf{R}, m, t, \gamma, \lambda, \alpha_1, \dots, \alpha_t, y, \varepsilon, j$ )  $\triangleright$  init  $j = 1$ 
2:   if  $j = t + 1$  then
3:     run the naive algorithm to compute  $\mathbf{C}$ , a list of correct pairs
4:   end if
5:   for  $\tilde{\Theta}(2^{y\alpha_j m})$  times do
6:      $A \leftarrow \text{partition}(\alpha_j m)$   $\triangleright$  random partition of size  $\frac{\alpha_j m}{2}$  of the  $\alpha_j m$  columns
7:      $\mathbf{L}' \leftarrow$  all  $\mathbf{v}_L \in \mathbf{L}$  with wt.  $H^{-1}(1 - \lambda)\frac{\alpha_j m}{2}$  on  $A$ -columns  $\triangleright$  naive search
8:      $\mathbf{R}' \leftarrow$  all  $\mathbf{v}_R \in \mathbf{R}$  with wt.  $H^{-1}(1 - \lambda)\frac{\alpha_j m}{2}$  on  $A$ -columns  $\triangleright$  naive search
9:     if  $|\mathbf{L}'|, |\mathbf{R}'|$  don't grow too large then  $\triangleright$  as defined in Lemma 3
10:     $\mathbf{C} \leftarrow \mathbf{C} \cup \text{NEARESTNEIGHBORREC}(\mathbf{L}', \mathbf{R}', m, t, \gamma, \lambda, \alpha_1, \dots, \alpha_t, y, \varepsilon, j + 1)$ 
11:     $\triangleright$  solve the problem recursively
12:    end if
13:   end for
14:   return  $\mathbf{C}$   $\triangleright$  output a list of correct pairs
15: end procedure

```

computation path and show that NEARESTNEIGHBOR has at least one of them with overwhelming probability over the coins of the algorithm. We continue by showing that the lists on that computation path achieve their expected size with overwhelming probability over the random choice of the input. We conclude with Theorem 1 that combines these results and shows the correctness and time complexity of NEARESTNEIGHBOR.

Lemma 1 (good distribution). *Let $(\mathbf{L}, \mathbf{R}, \gamma)$ be an instance of an (m, γ, λ) Nearest Neighbor problem with unknown solution vectors $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L} \times \mathbf{R}$. Let $\mathbf{z}^* := \mathbf{u}^* + \mathbf{v}^*$ and for any constant t let $\mathbf{u}^* := (\mathbf{u}_1^*, \dots, \mathbf{u}_t^*), \mathbf{v}^* := (\mathbf{v}_1^*, \dots, \mathbf{v}_t^*)$ $\mathbf{z}^* := (\mathbf{z}_1^*, \dots, \mathbf{z}_t^*)$ be a splitting of the vectors in t strips with sizes $\alpha_j m$ for all $1 \leq j \leq t$ with $\alpha_1 + \dots + \alpha_t = 1$. Then the double rerandomization of algorithm NEARESTNEIGHBOR guarantees with overwhelming probability that*

$$\mathbf{wt}(\mathbf{z}_j^*) = \gamma \alpha_j m \quad \text{and} \quad \mathbf{wt}(\mathbf{u}_j^*) = \mathbf{wt}(\mathbf{v}_j^*) = \frac{1}{2} \alpha_j m \quad \text{for all } 1 \leq j \leq t$$

in at least one of the rerandomized input lists.

Proof. In the first loop, random permutations π of the m columns are chosen. Thus the probability for $\mathbf{wt}(\mathbf{z}_j^*) = \gamma \alpha_j m$ for all $1 \leq j \leq t$ is

$$\binom{\alpha_1 m}{\gamma \alpha_1 m} \cdot \dots \cdot \binom{\alpha_t m}{\gamma \alpha_t m} / \binom{m}{\gamma m}.$$

A cancellation of the common terms, an application of Stirling's formula and the fact that t is constant shows the claim. In the second loop we choose uniformly random $\mathbf{r} = (\mathbf{r}_1, \dots, \mathbf{r}_t) \in \mathbb{F}_2^{\alpha_1 m} \times \dots \times \mathbb{F}_2^{\alpha_t m}$ with weight $\frac{1}{2} \alpha_j m$ on each of the t strips and add them to all elements in \mathbf{L} and \mathbf{R} . Fix one of the strips j and consider the vectors $(\mathbf{u}_j^*, \mathbf{v}_j^*)$. Let c_{01} denote the number of columns such that \mathbf{u}_j^* has a 0-coordinate and \mathbf{v}_j^* has a 1-coordinate. Define c_{00}, c_{10} and c_{11} analogously.

We define \mathbf{r} to be good on the strip j , if it has exactly $\frac{1}{2}c_{xy}$ ones in all four parts xy . The probability for that is

$$\binom{c_{00}}{\frac{1}{2}c_{00}} \binom{c_{01}}{\frac{1}{2}c_{01}} \binom{c_{10}}{\frac{1}{2}c_{10}} \binom{c_{11}}{\frac{1}{2}c_{11}} / \binom{\alpha_j m}{\frac{1}{2}\alpha_j m}.$$

Notice that this is again inverse polynomial, because $c_{00} + c_{01} + c_{10} + c_{11} = \alpha_j m$ per definition. Thus the probability stays polynomial for all t strips, since t is constant.

We conclude that a good \mathbf{r} solves the problem, because on each strip

$$\mathbf{wt}(\mathbf{u}_j^* + \mathbf{r}_j) = \mathbf{wt}(\mathbf{v}_j^* + \mathbf{r}_j) = \frac{1}{2}(c_{00} + c_{01} + c_{10} + c_{11}) = \frac{1}{2}\alpha_j m. \quad \square$$

In the following we use the notion of a *good* computation path inside the computation tree of our algorithm. See Fig. 7 for an example. In this figure the good path is marked as \rightarrow , whereas all the other paths are marked as dashed arrows.

Definition 3 (good computation path). Let $(\mathbf{u}^*, \mathbf{v}^*) \in \mathbf{L} \times \mathbf{R}$ be the target solution. A computation path of NEARESTNEIGHBOR is called *good*, if $(\mathbf{u}^*, \mathbf{v}^*)$ is contained in all t sublist pairs from the root to a leaf.

Lemma 2 (correctness). Let $t \in \mathbb{N}$ be the (constant) depth of NEARESTNEIGHBOR and $\lambda < 1 - H(\frac{\gamma}{2})$. Then the computation tree of NEARESTNEIGHBOR has a good computation path with overwhelming probability over the coins of the algorithm.

Proof. By construction, the target solution $(\mathbf{u}^*, \mathbf{v}^*)$ is contained in the initial list pair $\mathbf{L} \times \mathbf{R}$ on level 1 of the computation tree. In the following we show that if the solution is in one of the input lists on a level j , then with overwhelming probability it is also in one of the output lists on level j (which are either the input lists on level $j+1$ or the input lists for the naive algorithm on the last level). Thus, if this holds for any $1 \leq j \leq t$, we have a *good* path by induction.

Let us create $2^{y\alpha_j m}$ sublist pairs for each input list pair on level j with y from (5). On each level j we therefore have a total of $2^{y(\alpha_1 + \dots + \alpha_j)m}$ output list pairs, resulting in a total of 2^{ym} output list pairs on the last level.

Fix a level j and a target solution $(\mathbf{u}^*, \mathbf{v}^*)$ in one of the input pairs \mathbf{L}, \mathbf{R} on that level. On this level, we work on a strip of size $\alpha_j m$. Let \mathbf{u}_j^* and \mathbf{v}_j^* be the restrictions of \mathbf{u}^* , resp. \mathbf{v}^* to that strip. Due to the rerandomization in our algorithm, it is guaranteed by Lemma 1 that $\mathbf{wt}(\mathbf{u}_j^*) = \mathbf{wt}(\mathbf{v}_j^*) = \frac{1}{2}\alpha_j m$ and that their Hamming distance is $\Delta(\mathbf{u}_j^*, \mathbf{v}_j^*) = \gamma\alpha_j m$.

Thus, if we look at the pairwise coordinates of $(\mathbf{u}_j^*, \mathbf{v}_j^*)$ this implies that we have exactly $\frac{1-\gamma}{2}\alpha_j m$ $(0, 0)$ -pairs and $(1, 1)$ -pairs and exactly $\frac{\gamma}{2}\alpha_j m$ $(0, 1)$ -pairs and $(1, 0)$ -pairs, respectively. We illustrate this *input distribution* of the target pair $(\mathbf{u}_j^*, \mathbf{v}_j^*)$ in Fig. 8.

The algorithm constructs sublists \mathbf{L}', \mathbf{R}' by choosing a random partition A of the α_j -strip with $|A| = \frac{1}{2}\alpha_j m$. The algorithm only keeps those vectors of

$$\begin{aligned}\mathbf{u}_j^* &= 0 \dots 0 \ 0 \dots 0 \ 1 \dots 1 \ 1 \dots 1 \\ \mathbf{v}_j^* &= 0 \dots 0 \ 1 \dots 1 \ 0 \dots 0 \ 1 \dots 1 \\ \text{weight} & \frac{1-\gamma}{2} \alpha_j m \ \frac{\gamma}{2} \alpha_j m \ \frac{\gamma}{2} \alpha_j m \ \frac{1-\gamma}{2} \alpha_j m\end{aligned}$$

Fig. 8. input distribution ($\alpha_j m$ -strip)

the input lists that have a relative Hamming weight of $h := H^{-1}(1 - \lambda)$ on the columns defined by A , a choice that will be justified in Lemma 3. The choice of h implies that the number of $(1, 0)$ overlaps on the columns defined by A plus the number of $(1, 1)$ overlaps on the columns defined by A is $h\alpha_j \frac{m}{2}$. This is also the case for the number of $(0, 1)$ overlaps plus the number of $(1, 1)$ overlaps. Finally, we also know that the sum of all overlaps $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$ is $\alpha_j \frac{m}{2}$. Compared to the input distribution, this leaves one degree of freedom, which we denote by a parameter $0 \leq c \leq h$. We obtain the *output distribution* shown in Fig. 9.

$$\begin{aligned}\mathbf{u}_{j,A}^* &= 0 \dots 0 \ 0 \dots 0 \ 1 \dots 1 \ 1 \dots 1 \\ \mathbf{v}_{j,A}^* &= 0 \dots 0 \ 1 \dots 1 \ 0 \dots 0 \ 1 \dots 1 \\ \text{weight} & (1 - h - c) \alpha_j \frac{m}{2} \ c \alpha_j \frac{m}{2} \ c \alpha_j \frac{m}{2} \ (h - c) \alpha_j \frac{m}{2}\end{aligned}$$

Fig. 9. output distribution (A -columns of an $\alpha_j m$ -strip)

In order to compute the necessary number of repetitions, we have to compute the number of *good* partitions A that lead to an output distribution of Fig. 9 for any $0 \leq c \leq h$. This can be computed by multiplying the number of choices we have for each overlap of zeros and ones for any possible value of $c \alpha_j \frac{m}{2}$, which is

$$\sum_{c \alpha_j \frac{m}{2} = 0}^{h \alpha_j \frac{m}{2}} \binom{\frac{1-\gamma}{2} \alpha_j m}{(1 - h - c) \alpha_j \frac{m}{2}} \cdot \binom{\frac{\gamma}{2} \alpha_j m}{c \alpha_j \frac{m}{2}}^2 \cdot \binom{\frac{1-\gamma}{2} \alpha_j m}{(h - c) \alpha_j \frac{m}{2}}.$$

For a fixed c , it is for example possible to choose $c \alpha_j \frac{m}{2}$ $(0,1)$ overlaps in the A -area from an overall number of $\frac{\gamma}{2} \alpha_j m$ $(0,1)$'s in the whole strip.

Notice that some choices of c might lead to no possible choice for A , e.g. if $c > \gamma$. We determine a c that maximizes the number of *good* partitions. Numerical optimization shows that this maximum is obtained at $c = \frac{\gamma}{2}$. Notice that this is a valid choice for c (i.e. none of the binomial coefficients is zero) due to the restriction $\lambda < 1 - H(\frac{\gamma}{2})$ of Theorem 1 that implies $h > \frac{\gamma}{2}$. Thus the number of good partitions (up to polynomial factors) is

$$\binom{\frac{1-\gamma}{2} \alpha_j m}{(1 - h - \frac{\gamma}{2}) \alpha_j \frac{m}{2}} \cdot \binom{\frac{\gamma}{2} \alpha_j m}{\frac{\gamma}{2} \alpha_j \frac{m}{2}}^2 \cdot \binom{\frac{1-\gamma}{2} \alpha_j m}{(h - \frac{\gamma}{2}) \alpha_j \frac{m}{2}} = \tilde{\mathcal{O}} \left((2^{\alpha_j m})^{\gamma + (1-\gamma)H(\frac{h-\gamma}{1-\gamma})} \right)$$

The total number of partitions is $\binom{\alpha_j m}{\frac{1}{2}\alpha_j m} = \tilde{\mathcal{O}}(2^{\alpha_j m})$. Thus the expected number r of repetitions until we find the correct pair is the total number of partitions divided by the number of good partitions, which is $r = \tilde{\mathcal{O}}(2^{y\alpha_j m})$, which justifies our choice of y in identity (5).

It suffices to choose mr repetitions in order to find the correct pair with overwhelming probability, since the probability to *not* find the correct pair can be upper bounded by

$$(1 - 1/r)^{mr} \leq 2^{-m}.$$

Thus the probability that the algorithm goes wrong in *any* of its $2t$ calls on a *good* computation path can be upper bounded by $2t \cdot 2^{-m}$, which is negligible. Notice that $mr = \tilde{\mathcal{O}}(2^{y\alpha_j m})$, since polynomial factors vanish in $\tilde{\mathcal{O}}$ -notation. \square

Lemma 3 (list sizes). *Let $t \in \mathbb{N}$ be the (constant) depth of NEARESTNEIGHBOR and $\varepsilon > 0$. Consider a good computation path of depth t . Then with overwhelming probability the $2t$ lists inside the good computation path have sizes $\tilde{\mathcal{O}}((2^{\lambda m})^{1-\sum_{i=1}^j \alpha_i + \frac{\varepsilon}{2}})$ for all $1 \leq j \leq t$ and thus are not cut off by the algorithm.*

Proof. Fix some $1 \leq j \leq t$ and an initial input list \mathbf{L} with $|\mathbf{L}| = 2^{\lambda m}$ (the argument is analogous for \mathbf{R}). For each vector $\mathbf{v}_k \in \mathbf{L}$ we define random variables X_k such that

$$X_k = \begin{cases} 1 & \text{if } \bigwedge_{i=1}^j \mathbf{v}_k \in \mathbf{L}^i \\ 0 & \text{otherwise} \end{cases}.$$

Let $X := \sum_{k=1}^{|\mathbf{L}|} X_k$. Thus the random variable X counts the number of elements in the output list \mathbf{L}^j . Recall that NEARESTNEIGHBOR restricts to relative weight $h = H^{-1}(1 - \lambda)$ on the columns in A . Since we know that the computation path is *good*, there is one $\mathbf{v}_{k^*} \in \mathbf{L}$ with $\mathbb{P}[X_{k^*} = 1] = 1$. Notice that all the other elements are independent of \mathbf{v}_{k^*} and are uniformly chosen among all vectors with weight $\alpha_j \frac{m}{2}$ on each strip j . Thus for all $\mathbf{v}_k \in \mathbf{L} \setminus \{\mathbf{v}_{k^*}\}$ we have

$$\mathbb{P}[X_k = 1] = \prod_{i=1}^j \left(h \frac{\alpha_i m}{2} \right) \left((1 - h) \frac{\alpha_i m}{2} \right) / \binom{\alpha_i m}{\frac{\alpha_i m}{2}},$$

which is, for each of the first j strips, the number of vectors that have relative weight h on the A -columns divided by the number of all possible vectors. Thus the expected size of the output list is

$$\mathbb{E}[X] = 1 + (2^{\lambda m} - 1) \cdot \prod_{i=1}^j \left(h \frac{\alpha_i m}{2} \right) \left((1 - h) \frac{\alpha_i m}{2} \right) / \binom{\alpha_i m}{\frac{\alpha_i m}{2}}.$$

Notice that obviously $\mathbb{E}[X] \geq 1$. Applying Chebyshev's inequality, we get

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq 2^{\frac{\varepsilon}{2} m} \mathbb{E}[X]] \leq \frac{\mathbb{V}[X]}{2^{\varepsilon m} \mathbb{E}[X]^2} \leq \frac{1}{2^{\varepsilon m} \mathbb{E}[X]} \leq 2^{-\varepsilon m},$$

using $\mathbb{V}[X] = \mathbb{V}[\sum_k X_k] = \sum_k \mathbb{V}[X_k] = \sum_k (\mathbb{E}[X_k^2] - \mathbb{E}[X_k]^2) \leq \sum_k \mathbb{E}[X_k] = \mathbb{E}[X]$. We have $\mathbb{V}[\sum_k X_k] = \sum_k \mathbb{V}[X_k]$, because the X_k are pairwise independent.

Thus (for both lists on each level) we obtain $\mathbb{P}[X \text{ too large in any of the steps}] \leq 2t \cdot 2^{-\varepsilon m}$, applying the union bound.

From Stirling's formula it also follows that $\mathbb{E}[X] \leq (2^{\lambda m})^{1-\sum_{i=1}^j \alpha_i}$ for each $1 \leq j \leq t$. Hence, with overwhelming probability, the list sizes are as claimed. \square

Now we are able to prove Theorem 1.

Theorem 1. *For any constant $\varepsilon > 0$ and any $\lambda < 1 - H(\frac{\gamma}{2})$, NEARESTNEIGHBOR solves the (m, γ, λ) NN problem with overwhelming probability (over both the coins of the algorithm and the random choice of the input) in time*

$$\tilde{\mathcal{O}}\left(2^{(y+\varepsilon)m}\right) \quad \text{with} \quad y := (1-\gamma)\left(1 - H\left(\frac{H^{-1}(1-\lambda) - \frac{\gamma}{2}}{1-\gamma}\right)\right).$$

Proof. By Lemma 2, there is a path that includes the solution (with overwhelming probability over the coins of the algorithm). Fix that path. We show that by Lemma 3 (with overwhelming probability over the random choice of the input) NEARESTNEIGHBOR's time complexity is the maximum of the times

$$\tilde{\mathcal{O}}\left((2^m)^{\lambda(1-\sum_{i=1}^{j-1} \alpha_i) + y \sum_{i=1}^j \alpha_i + \frac{\varepsilon}{2}}\right) \quad (6)$$

to create all sublists on level j for all levels $1 \leq j \leq t$ and the time

$$\tilde{\mathcal{O}}(2^{(y+\varepsilon)m}) \quad (7)$$

to naively solve the problem on the last level.

From Lemma 3 we know that on each level j the sizes of the *input* lists are $\tilde{\mathcal{O}}((2^m)^{\lambda(1-\sum_{i=1}^{j-1} \alpha_i) + \frac{\varepsilon}{2}})$. Notice that if the lists grow too large, we simply abort. On level j we construct $\tilde{\mathcal{O}}(2^{y\alpha_j m})$ new sublists for each input list so that we have a total number of $\tilde{\mathcal{O}}((2^m)^{\sum_{i=1}^j \alpha_i})$ sublists on this level. Each of these sublists is computed by naively searching through the input lists. Thus, we have to multiply the sizes of the input lists with the number of sublists which results in complexity (6).

In NEARESTNEIGHBOR's last step we use the naive algorithm to join a total number of $\tilde{\mathcal{O}}(2^{ym})$ pairs of sublists of size $\tilde{\mathcal{O}}(2^{\frac{\varepsilon}{2}m})$ each, resulting in complexity (7). The overall complexity is the maximum of complexities (6) and (7).

We want to continue by computing the overall time complexity of the steps defined by (6) for any fixed t by setting the complexities of (6) equal. Thus we choose

$$\lambda \left(1 - \sum_{i=1}^{j-1} \alpha_i\right) + y \sum_{i=1}^j \alpha_i + \frac{\varepsilon}{2} = \lambda \left(1 - \sum_{i=1}^j \alpha_i\right) + y \sum_{i=1}^{j+1} \alpha_i + \frac{\varepsilon}{2}$$

for all $1 \leq j \leq t-1$, which implies $\alpha_{j+1} = (\lambda/y) \cdot \alpha_j$. Additionally, we need $\sum_{i=1}^t \alpha_i = 1$, thus using $y > \lambda$ from Eq. (5) we get $1 = \sum_{i=1}^t \alpha_i = \alpha_1 \cdot \sum_{i=0}^{t-1} (\lambda/y)^i = \alpha_1 \cdot \frac{1-(\lambda/y)^t}{1-(\lambda/y)}$. This implies $\alpha_1 = \frac{1-(\lambda/y)}{1-(\lambda/y)^t}$ and finally (uniquely)

determines all $(\alpha_1, \dots, \alpha_t)$. Since by our choice all complexities (6) are the same, we get an overall running time of $\tilde{\mathcal{O}}((2^m)^{\lambda+y\cdot\alpha_1+\frac{\varepsilon}{2}})$.

Notice that the (constant) choice $t = \left\lceil \frac{\log(y - \lambda + \frac{\varepsilon}{2}) - \log(\frac{\varepsilon}{2})}{\log(y) - \log(\lambda)} \right\rceil \in \mathbb{N}$ leads to $\alpha_1 = \frac{y - \lambda + \frac{\varepsilon}{2}}{y}$ and we finally obtain the same complexity $\tilde{\mathcal{O}}(2^{(y+\varepsilon)m})$ as in (7), which makes (7) the time complexity of the whole algorithm.

We want to conclude the proof by showing that for any fixed t it is indeed optimal to set all time complexities in (6) equal. Let $(\alpha_1, \dots, \alpha_t)$ be the (unique) choice defined above. Assume this is not optimal, thus a choice $(\tilde{\alpha}_1, \dots, \tilde{\alpha}_t)$ that is different from the first one improves upon the overall time complexity. Decreasing one of the time complexities implies that there is a $1 \leq k \leq t$ with $\tilde{\alpha}_k < \alpha_k$, because $y > \lambda$. Let k be minimal with that property.

Case 1: There is an $1 \leq \ell < k$ s.t. $\tilde{\alpha}_\ell > \alpha_\ell$ and let ℓ be minimal with that property. Then $\sum_{i=1}^{\ell-1} \tilde{\alpha}_i = \sum_{i=1}^{\ell-1} \alpha_i$. Thus $\lambda + (y - \lambda) \cdot \sum_{i=1}^{\ell-1} \tilde{\alpha}_i + y \cdot \tilde{\alpha}_\ell + \frac{\varepsilon}{2} > \lambda + (y - \lambda) \cdot \sum_{i=1}^{\ell-1} \alpha_i + y \cdot \alpha_\ell + \frac{\varepsilon}{2}$.

Case 2: Otherwise, we know that $\sum_{i=1}^{k-1} \tilde{\alpha}_i = \sum_{i=1}^{k-1} \alpha_i$ and thus $\sum_{i=1}^k \tilde{\alpha}_i < \sum_{i=1}^k \alpha_i$. Notice that it also has to hold that $\sum_{i=1}^t \tilde{\alpha}_i = \sum_{i=1}^t \alpha_i = 1$. Hence there is an $k < \ell \leq t$ s.t. $\sum_{i=1}^{\ell-1} \tilde{\alpha}_i < \sum_{i=1}^{\ell-1} \alpha_i$ and $\sum_{i=1}^\ell \tilde{\alpha}_i \geq \sum_{i=1}^\ell \alpha_i$. Thus $\lambda - \lambda \cdot \sum_{i=1}^{\ell-1} \tilde{\alpha}_i + y \cdot \sum_{i=1}^\ell \tilde{\alpha}_i > \lambda - \lambda \cdot \sum_{i=1}^{\ell-1} \alpha_i + y \cdot \sum_{i=1}^\ell \alpha_i$.

In both cases the time complexity on some level $\ell \neq k$ (and therefore the overall time complexity) strictly increases, which makes the new choice inferior to the original one. \square

Acknowledgments. We would like to thank the anonymous Eurocrypt reviewers for their helpful and detailed comments that improved and clarified our work.

Open Problem

Our NEARESTNEIGHBOR algorithm uses a recursion tree of constant depth t . This leads to a large *polynomial* blow-up for our decoding algorithm (in the size of m^t), which asymptotically vanishes but in practice might lead to an undesirably large break-even point with the BJMM algorithm. We pose it as an open problem to get rid of this polynomial overhead.

References

1. Agarwal, P.K., Edelsbrunner, H., Schwarzkopf, O.: Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs. *Discrete & Computational Geometry* **6**, 407–422 (1991)
2. Alekhnovich, M.: More on Average Case vs Approximation Complexity. In: 44th Symposium on Foundations of Computer Science (FOCS), pp. 298–307 (2003)
3. Becker, A., Coron, J.-S., Joux, A.: Improved Generic Algorithms for Hard Knapsacks. In: Paterson, K.G. (ed.) *EUROCRYPT 2011*. LNCS, vol. 6632, pp. 364–385. Springer, Heidelberg (2011)

4. 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)
5. Brakerski, Z., Vaikuntanathan, V.: Efficient Fully Homomorphic Encryption from (Standard) LWE. In: FOCS, pp. 97–106 (2011)
6. Dubiner, M., Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. IEEE Transactions on Information Theory 56(8), 4166–4179 (2010)
7. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC, pp. 197–206 (2008)
8. Har-Peled, S., Indyk, P.: Rajeev Motwani Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. Theory of Computing 8(1), 321–350 (2012)
9. Hopper, N.J., Blum, M.: Secure Human Identification Protocols. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 52–66. Springer, Heidelberg (2001)
10. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. J. ACM 21(2), 277–292 (1974)
11. Howgrave-Graham, N., Joux, A.: New Generic Algorithms for Hard Knapsacks. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 235–256. Springer, Heidelberg (2010)
12. Kiltz, E., Pietrzak, K., Cash, D., Jain, A., Venturi, D.: Efficient Authentication from Hard Learning Problems. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 7–26. Springer, Heidelberg (2011)
13. Lee, P.J., Brickell, E.F.: An Observation on the Security of McEliece’s Public-Key Cryptosystem. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 275–280. Springer, Heidelberg (1988)
14. 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)
15. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. Jet Propulsion Laboratory DSN Progress Report 42–44, 114–116 (1978)
16. Peikert, C.: Brent Waters Lossy trapdoor functions and their applications. In: STOC, pp. 187–196 (2008)
17. Peters, C.: Information-Set Decoding for Linear Codes over \mathbb{F}_q . In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 81–94. Springer, Heidelberg (2010)
18. Prange, E.: The Use of Information Sets in Decoding Cyclic Codes. IRE Transaction on Information Theory 8(5), 5–9 (1962)
19. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC, pp. 84–93 (2005)
20. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM 56(6) (2009)
21. Stern, J.: A method for finding codewords of small weight. In: Proceedings of the 3rd International Colloquium on Coding Theory and Applications, London, UK, pp. 106–113. Springer (1989)
22. Sudan, M.: Algorithmic Introduction to Coding Theory. Lecture Notes (available online) (2001)
23. Andoni, A., Indyk, P., Nguyen, H.L., Razenshteyn, I.: Beyond Locality-Sensitive Hashing. In: SODA, pp. 1018–1028 (2014)
24. Valiant, G.: Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas. In: FOCS, pp. 11–20 (2012)

Symmetric Cryptanalysis I

Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE

Itai Dinur^(✉)

Département d’Informatique, École Normale Supérieure, Paris, France
dinur@di.ens.fr

Abstract. The FX-construction was proposed in 1996 by Kilian and Rogaway as a generalization of the DESX scheme. The construction increases the security of an n -bit core block cipher with a κ -bit key by using two additional n -bit masking keys. Recently, several concrete instances of the FX-construction were proposed, including PRINCE (proposed at Asiacrypt 2012) and PRIDE (proposed at CRYPTO 2014). These ciphers have $n = \kappa = 64$, and are proven to guarantee about $127 - d$ bits of security, assuming that their core ciphers are ideal, and the adversary can obtain at most 2^d data.

In this paper, we devise new cryptanalytic time-memory-data tradeoff attacks on FX-constructions. While our attacks do not contradict the security proof of PRINCE and PRIDE, nor pose an immediate threat to their users, some specific choices of tradeoff parameters demonstrate that the security margin of the ciphers against practical attacks is smaller than expected. Our techniques combine a special form of time-memory-data tradeoffs, typically applied to stream ciphers, with recent analysis of FX-constructions by Fouque, Joux and Mavromati.

Keywords: Cryptanalysis · Block cipher · Time-memory-data tradeoff · FX-construction · DESX · PRINCE · PRIDE

1 Introduction

The Advanced Encryption Standard (AES) is the most widely used block cipher today. It is believed to guarantee a large security margin against practical attacks, and can therefore be used to encrypt very sensitive data. The AES was preceded by the Data Encryption Standard (DES), whose 56-bit key made it vulnerable to straightforward exhaustive search. Consequently, in 1984, when DES was still widely used, Ron Rivest proposed a simple solution (known as DESX [21]) to address the concern regarding its small key size. The DESX construction simply XORs two independent 64-bit keys at the beginning and at the end of the core DES encryption process, such that the total key size becomes $56 + 64 + 64 = 174$ bits. This construction was generalized to the so-called *FX-construction* by Kilian and

Rogaway in 1996 [18]. The FX-construction is built using an (arbitrary) n -bit block cipher F_K with a κ -bit key K and two additional n -bit whitening keys K_1, K_2 , and defined as $FX_{K, K_1, K_2}(P) = K_2 \oplus F_K(K_1 \oplus P)$. Kilian and Rogaway proved that the cipher guarantees $\kappa + n - d - 1$ bits of security,¹ assuming that F is a perfect block cipher and the adversary can obtain $D = 2^d$ plaintext-ciphertext pairs. Furthermore, Kilian and Rogaway showed that the bound is tight by extending the attack of Daemen [10] (on the related Even-Mansour construction) to a simple attack on the FX-construction with complexity of about $2^{\kappa+n-d-1}$.

The analysis of Kilian and Rogaway implies that the security of the FX-construction depends on how much data the attacker can obtain, and thus the security is not completely determined by the computational power of the attacker. This is a unique situation, as for (almost) all block ciphers used in practice today that have no known weaknesses, obtaining additional data does not seem to give any significant advantage in key recovery attacks. Thus, the security level of $\kappa + n - d - 1$ does not allow to directly compare FX-constructions to classical ciphers, and does not give a clear indication on the effort required in order to break such a construction.

Until recently, the security guaranteed by FX-constructions was perhaps not very relevant, as such constructions were not proposed for practical use (apart from DESX). This situation changed in 2012, when the FX-construction PRINCE was presented at Asiacrypt [7], and more recently, at CRYPTO 2014, a similar FX-construction (named PRIDE [1]) was proposed.² Both of these constructions have $n = \kappa = 64$, and thus they offer security of about $127 - d$ bits, assuming that their core ciphers are ideal.³

In order to encourage its adoption by the industry, the designers of PRINCE launched a competition (named the PRINCE Challenge [23]), calling for cryptanalysis of the cipher which would lead to better understanding of its security. The competition focuses on practical attacks on round-reduced variants of PRINCE, where a practical attack is defined to have data complexity of (up to) 2^{30} known plaintexts (or 2^{20} chosen plaintexts), time complexity of 2^{64} and memory complexity of 2^{45} bytes.

Motivated by the PRINCE Challenge, in this paper, we investigate the security margin guaranteed by FX-constructions against practical attacks, with a focus on PRINCE and PRIDE (i.e., FX-constructions with $n = \kappa = 64$). We first analyze well-known generic attacks on FX-constructions [5, 12, 18], and conclude that these attacks do not threaten the security of PRINCE and PRIDE. Then, we devise new attacks with lower memory complexity, and claim that the

¹ A cipher guarantees b bits of security if the complexity of the most efficient attack on it is at least 2^b .

² PRINCE and PRIDE are FX-constructions of a particular type, where K_2 linearly depends on K_1 . However, it is shown in [7] that the smaller key size does not reduce the security of the schemes against generic attacks.

³ PRINCE guarantees slightly less than $127 - d$ bits of security, as its core cipher was designed to preserve a special property that ensures a small footprint.

security margin of FX-constructions with $n = \kappa = 64$ against these attacks is somewhat reduced (although the attacks remain impractical).

Despite the new attacks described above, our most interesting attacks are carried out in Hellman's time-memory tradeoff model [16]. In this model, the adversary spends a lot of resources on a one-time preprocessing phase that analyzes the scheme, and whose output is stored in (relatively small) memory. After this one-time preprocessing phase is completed, the scheme can be attacked much more efficiently, and this makes Hellman's model attractive in many cases.

The starting point of our attacks is a recent analysis of the FX-construction and related designs by Fouque et al. [14]. One of the attacks of [14] on PRINCE has data complexity of 2^{32} and a very efficient time complexity of 2^{32} . The main shortcomings of this attack are its huge memory complexity of about 2^{67} bytes, and its impractical preprocessing phase, which has time complexity of 2^{96} . The techniques we develop trade off these high memory and preprocessing complexities with time and data complexities, and allow to obtain more balanced and practical tradeoffs.

Some concrete parameters of our attacks on PRICE and PRIDE in Hellman's model are summarized in Table 1. Consider the online phase of Attack 1, which requires about 2^{32} data, takes 2^{64} time and requires 2^{51} bytes of memory. The parameters of this attack are thus not far from the parameters considered in the PRINCE challenge [23] as practical, and they are valid regardless of the cipher's internal number of rounds. Furthermore, we show in this paper that Attack 1 (as well as our other online attacks in Table 1) rarely accesses the memory (which can be stored on a hard disk), and can be efficiently realized using dedicated hardware with a budget of a medium-size enterprise. Therefore, we consider this attack to be semi-practical.

Attack 1 has two main shortcomings: it requires the 2^{32} data in the form of adaptively chosen plaintexts, and more significantly, it requires a long and impractical one-time precomputation phase of⁴ complexity 2^{96} .

In order to reduce the preprocessing complexity, we consider Attack 2 which exploits a larger number of 2^{40} adaptively chosen plaintexts. This data can be collected (for example) if the attacker can obtain black-box access to the encryption device for a few hours, and can thus be considered practical in some (restricted) scenarios. The online attack runs in time 2^{56} , requires 2^{51} bytes of storage, and is therefore even more efficient than the online phase of Attack 1. More significantly, it requires a shorter precomputation phase of time complexity 2^{88} , which is still impractical, but only marginally.⁵

⁴ Note that according to the bound of [18], any generic attack on FX-constructions with $n = \kappa = 64$ using 2^{32} data, must have time complexity of at least $2^{64+64-32-1} = 2^{95}$.

⁵ We assume that some adversaries can spend a huge amount of resources on preprocessing (in contrast to online attacks). Therefore, we consider preprocessing time complexity of 2^{80} to be (marginally) practical, as demonstrated by the capacity of the Bitcoin network [6], and supported by the NIST recommendation to disallow 80-bit keys after 2014 [20].

An interesting observation is that when we execute a $0 < p \leq 1$ fraction of the preprocessing phase of Attack 2, then the key recovery attack succeeds with probability p . If we consider $p = 2^{-8}$, the attack succeeds with a non-negligible probability of $p \approx 1/256$, requires only $2^{51-8} = 2^{43}$ bytes of disk space (or 8 terabytes), and can be implemented today with a small academical budget (similar tasks have been implemented with such a budget [15]). Moreover, the preprocessing time complexity of the attack above becomes $2^{88-8} = 2^{80}$ (which is more practical than 2^{88}). This shows that it may be beneficial to start the preprocessing phase today, instead of waiting for the technology that would make it fully realizable in the future. We further note that the complexity parameters of Attack 2 for $p = 2^{-8}$ and $n = \kappa = 64$ are, in fact, equivalent to those for $p \approx 1$ and $n = 64, \kappa = 56$. Since DES has a 56-bit key, the full attack against DESX could potentially be carried out today by a resourceful adversary.

Table 1. Attacks on PRINCE and PRIDE

Attack ID	Reference	Data (ACP) [†]	Preprocessing Time	Online Time	Memory (Bytes)	Online Attack Cost Estimate ^{††} (US Dollars)
—	[14]	2^{32}	2^{96}	2^{32}	2^{67}	$> 10,000,000,000$
1	This paper	2^{32}	2^{96}	2^{64}	2^{51}	$< 1,000,000$
2	This paper	2^{40}	2^{88}	2^{56}	2^{51}	$< 1,000,000$
3	This paper	2^{40}	2^{88}	2^{64}	2^{47}	$< 1,000,000$
4	This paper	2^{48}	2^{80}	2^{64}	2^{51}	$< 1,000,000$

[†] Adaptively chosen plaintexts

^{††} As estimated at the end of Section 3

Although the FX-construction is a block cipher, our techniques are borrowed from cryptanalysis of stateful ciphers (i.e., stream ciphers). We first notice that the FX-construction can be viewed as a (standard) core block cipher with an additional secret state (namely, the input or output to the core block cipher), which is hidden by the masking keys using simple XOR operations. Our main methodological contribution is to use Hellman’s time-memory tradeoff to invert a set of special states, similarly to the techniques that Biryukov, Shamir and Wagner applied to stream ciphers which have low sampling resistance [3,4]. However, unlike the case of stream ciphers, in some cases we analyze (in particular for $d > n/2$), we have to request the data and optimize our algorithms in a non-trivial way in order to obtain efficient tradeoffs.

A unique feature of our time-memory-data tradeoff curve for $2^d \leq 2^{n/2}$, is that the effective hidden state (key) size of the FX-construction is reduced by a factor of $2^{1.5d}$ in the online phase of the attack. On the other hand, for stream ciphers, the effective hidden state size is only reduced by a factor of 2^d . The reason for this is that in most stream ciphers, the hidden state is permuted and its entropy is maintained when producing keystream. On the other hand, we exploit the basic technique of Fouque et al., which applies a non-bijective function

to the hidden state of the FX-construction, reducing its entropy. In particular, for $\kappa = n = 64$ and $2^d = 2^{32}$, the effective key size is reduced to $64 + 64 - 1.5 \cdot 32 = 80$ bits, whereas one could have expected it to be $64 + 64 - 32 = 96$ bits. Exploiting memory to further reduce the effective key size, leads to online key recovery attacks on PRINCE and PRIDE with semi-practical complexities.

The paper is organized as follows. We begin by introducing our notation in Section 2, while Section 3 provides an overview of our attacks. Section 4 gives some necessary background, and our simple attacks (that do not use a preprocessing phase) are described in Section 5, while our advanced attacks are described in Section 6. Finally, we conclude the paper in Section 7.

2 Notations and Conventions

The FX-construction [18] is built using an (arbitrary) n -bit block cipher F_K with a κ -bit key K and 2 additional n -bit whitening keys K_1, K_2 , and defined as $FX_{K, K_1, K_2}(P) = K_2 \oplus F_K(K_1 \oplus P)$. We denote the plaintext by P , its ciphertext $FX_{K, K_1, K_2}(P) = K_2 \oplus F_K(K_1 \oplus P)$ by C , and the inner values $K_1 \oplus P$ and $F_K(K_1 \oplus P)$ by X and Y , respectively (see Figure 1).

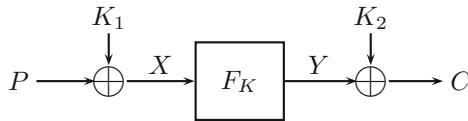


Fig. 1. The FX-Construction

In this paper, we are also interested in more specific instances of the FX-construction. In particular, this construction also inspired the design of the Even-Mansour scheme [13], in which the core cipher is, in fact, an unkeyed public permutation F (for which $\kappa = 0$). Furthermore, a major focus of this paper is placed on the recently proposed concrete FX-constructions PRINCE [7] and PRIDE [1]. These constructions use only n bits of whitening key material K_1 , where K_2 is defined by $A(K_1)$, for an invertible affine function A . We refer to this simplified scheme as an *SFX-construction*.

As we deal with various tradeoffs between complexity parameters of attacks, we define some notation that is used in order to quantify these parameters. For an n -bit block cipher, we denote $N = 2^n$. When considering an attack, we denote by T its total online time complexity, where a unit of time corresponds to an encryption of a single plaintext. We denote by $M = 2^m$ the memory complexity of the attack in terms of n -bit words, and by $D = 2^d$ its data complexity in terms of plaintext-ciphertext pairs. Finally, we denote by \hat{T} the preprocessing time complexity of the attack (if the attack does not require preprocessing, then $\hat{T} = 0$).

The online and preprocessing time complexities of our attacks throughout this paper are formulated in terms of the parameters n, κ, d, m defined above. For the sake of convenience, we assume in several parts of this paper that $\kappa = n$, which is the case for PRINCE and PRIDE. We note that if $\kappa > n$, the attacks we describe have a small penalty of about $\lceil \kappa/n \rceil$ in the time and memory complexities.⁶

Since we estimate the practicality of some of our attacks, it is insufficient to merely compute their time, data, and memory complexities. Indeed, the practicality of an attack is largely influenced by more subtle properties such as the number of memory lookups during its execution (which determines whether the memory has to be stored in RAM, or can be stored on a cheaper hard disk), and whether the workload of the attack can be easily parallelized (i.e., divided across different CPUs). Another crucial element is the size of the implementation circuit, which determines whether the attack can be efficiently realized on cheap dedicated hardware.

3 Overview of Previous and New Attacks on FX-Constructions

The new and previously published tradeoffs for FX-constructions are summarized in Table 2. We now compare these attacks at a high level, and then emphasize their practicality for $n = \kappa = 64$, focusing on the concrete parameters for the attacks given in Table 1 that use preprocessing.⁷

Attacks without Preprocessing. We first examine attacks with no preprocessing, for which an initial chosen plaintext attack was described in [18]. Then, a known plaintext attack with the same complexity was published in [5] for $D = 2^{n/2}$, and later generalized in [12] to work with any number of known plaintexts. All of these attacks require $M = D$ memory and seem impractical for $n = \kappa = 64$, as they either require impractical data and memory (e.g. for $D = M = 2^{64}$), or impractical time (e.g., for $D = M = 2^{32}$ then $T = 2^{96}$). Intermediate values such as $D = M = 2^{48}$ and $T = 2^{80}$ may seem more practical, but we note that the large memory of 2^{48} has to be accessed a huge number of 2^{80} times (essentially, for each cipher evaluation). While this can be somewhat optimized by grouping together the memory lookups, these parameters still seem completely impractical.

The new attacks we describe in Section 5 show that in the adaptively chosen plaintext model, we can mount attacks with the same data and time complexities, and a reduced memory complexity. In particular, for $D = 2^{n/2}$, our attack requires negligible memory, while the attacks of [5, 12, 18] require $2^{n/2}$ memory. For $n = \kappa = 64$ and $D = 2^{48}$, our attack requires only $2^{2 \cdot d - n} = 2^{2 \cdot 48 - 64} = 2^{32}$

⁶ The ratio between the key size and block size is typically small in modern block ciphers.

⁷ We note that optimal choice of parameters and infrastructure to realize an attack depends of the setting, and there are many more options than listed in Table 1.

words of memory, which is significantly better than the attack of [12]. In this case, the time complexity is 2^{80} , which can be considered as marginally practical, but is still a huge effort put into recovering a single key. Furthermore, as the data collection cannot be parallelized, obtaining 2^{48} data is generally not considered practical. Nevertheless, we still believe that the attack for $D = 2^{48}$ serves as an initial indication that the security margin of PRINCE and PRIDE is smaller than expected.

Attacks with Preprocessing. We examine the attacks that require preprocessing assuming $2^d \leq 2^{n/2}$, for which the previously published attack was given in [14]. This attack has a very efficient online time complexity of 2^d , but requires $2^{\kappa+n-2d} \geq 2^\kappa$ words of memory. Our attack trades-off this memory at the expense of increasing the time complexity, and obtains $T = 2^{2(\kappa+n-m-1.5d)}$.

For $d > n/2$, assuming $2^m \leq 2^{\kappa+n-2d}$, we can further reduce the preprocessing complexity and obtain a more efficient online time complexity of $2^{2(\kappa+n/2-m-d/2)}$. Concrete parameter sets for ciphers with $n = \kappa = 64$ are given in Table 1.

The Practicality of Our Attacks. As we show in sections 6.1 and 6.2, the online attacks summarized in Table 1 rarely access the memory, which can be stored on a hard disk. Moreover, the attacks can be easily parallelized, and although they are conceptually non-trivial, their circuit sizes are almost as small as the circuits of the attacked FX-constructions. Therefore, the attacks can be efficiently implemented on dedicated hardware.

As a consequence of the above, we estimate that the online phase of the attacks in Table 1 (which require at most 2^{64} cipher evaluations and 2^{51} bytes of storage) can be realized today by a medium-size enterprise with a budget of several hundred thousand dollars. This rough estimation is based on the fact that up to about 2^{64} cipher evaluations can be performed in a few weeks on dedicated hardware with such a budget [9]. Considering storage, a standard 1-terabyte hard disk costs about 100 US dollars (as of 2015). Therefore, 2^{51} bytes of storage (or about 2,000 terabytes) cost roughly 200,000 dollars. Of course, fully realizing an attack requires additional expenses, but we do not expect them to increase the overall cost by a significant factor. On the other hand, using the same metric, we estimate the cost of 2^{67} bytes of storage to be more than 10,000,000,000 US dollars, making the attack of [14] more expensive than our attacks by a factor larger than 10,000.

4 Background

The new attacks described in this paper combine several previously published techniques, which are described in this section.

Table 2. Time-Memory-Data Tradeoffs for FX-Constructions

Reference	Data	Preprocessing Time	Online Time	Memory
[12]	$2^d \leq 2^n$ KP [†]	-	$2^{\kappa+n-d}$	2^d
Section 5.1	$2^d \leq 2^{n/2}$ ACP ^{††}	-	$2^{\kappa+n-d}$	negligible
Section 5.2	$2^d > 2^{n/2}$ ACP ^{††}	-	$2^{\kappa+n-d}$	2^{2d-n}
[14]	$2^d \leq 2^{n/2}$ ACP ^{††}	$2^{\kappa+n-d}$	2^d	$2^{\kappa+n-2d}$
Section 6.1	$2^d \leq 2^{n/2}$ ACP ^{††}	$2^{\kappa+n-d}$	$2^{2(\kappa+n-m-1.5d)}$	2^m
Section 6.2	$2^d > 2^{n/2}$ ACP ^{††}	$2^{\kappa+n-d}$	$2^{2(\kappa+n/2-m-d/2)}$	$2^m \leq 2^{\kappa+n-2d}$
Section 6.2	$2^d > 2^{n/2}$ ACP ^{††}	$2^{\kappa+n-d}$	$2^{\kappa+d-m}$	$2^m > 2^{\kappa+n-2d}$

[†] Known plaintexts

^{††} Adaptively chosen plaintexts

4.1 Hellman’s Time-Memory Tradeoff [16] (with Preprocessing)

We summarize Hellman’s classical time-memory tradeoff attack [16] on an n -bit block cipher E_K , assuming that $\kappa = n$. In the preprocessing phase, we fix a plaintext P , and define the function $h(\{0, 1\}^n) \rightarrow \{0, 1\}^n$ as $h(K) = E_K(P)$. The goal in this phase is to cover most (more than half) of the key space with chains defined by iterating the function h . For parameters M' and T' , we choose M' arbitrary starting points for the chains, where each chain is of length T' . We store in a table only the (startpoint, endpoint) pair⁸ of each chain and sort the table according to the endpoint value. Such a table requires M' words of memory, and is referred to as a *Hellman table*.

After evaluating M' chains, and reaching the birthday bound (stopping rule) of $T' \cdot M' T' = N$, adding additional chains to the table is wasteful. Thus, we can cover $M' T' = N/T'$ points with M' words of memory. In order to cover most of the key space, we use flavors of h , where flavor i is defined (for example) as $h^{[i]}(K) = h(K) + i$. Thus, we use T' flavors of h , and compute a Hellman table for each flavor, covering a total of about $N/T' \cdot T' = N = 2^\kappa$ keys as required. The T' tables are the output of the preprocessing phase, and they require $M = M' T'$ words of memory, and a total of $\hat{T} = N$ computation time.

During the online phase, we request the encryption of P under the unknown key K , $E_K(P)$. In order to recover K , we try to invert it using each of the Hellman tables by iteratively calculating $h^{[i]}$ starting from $E_K(P)$, and searching if the current value is an endpoint in the table. Once we reach an endpoint, we obtain its startpoint, and continue the evaluation, hoping to reach $(h^{[i]})^{-1}(E_k(P) + i)$ and to recover K . This process has a time complexity of about T' for each Hellman table. Thus, the total online time complexity is $T = T'^2$, and as $M = M' T'$ and $M' T'^2 = N$, we obtain a time-memory tradeoff of $TM^2 = N^2$, or $T = 2^{2(\kappa-m)}$.

⁸ We note that we can save a large fraction of the memory required for startpoint storage by exploiting the freedom to choose them, as described in [2]. Thus, we assume that a chain requires a single word of storage.

Reducing memory lookups using distinguished points. A simple variant of Hellman's algorithm (attributed to Ron Rivest, and later analyzed in [8, 22]), stops each chain once it reaches a set of *distinguished points*, which are defined according to an easily verifiable condition on $h^{[i]}(K)$. For example, in case we require chains of length T' , we define the set of distinguished points to contain the points whose $\log(T')$ LSBs are zero. With this variant, the length of the chains is variable and is only defined on average, but this does not result in a significant penalty on the theoretical time complexity of the attack. On the other hand, the distinguished points method has a big advantage in practice, as we only need to access a large Hellman table once for (about) every T' evaluations of $h^{[i]}$ in the online phase of the attack. The small number of memory lookups allows the attacker to store the memory on hard disk, which is much cheaper than RAM.

Parallelization. Since each of the T' Hellman tables can be searched independently, the computation can be divided across (at most) T' CPUs, each requiring access to a Hellman table of size M' . Furthermore, each CPU is expected to access the memory only once during the computation of T' time (in order to find a startpoint that corresponds to an endpoint). Consequently, time-memory tradeoff algorithms can be implemented relatively cheaply on dedicated hardware [19, 22].

For example, in case where $\kappa = 64$ and we have $M = 2^{48}$ available words of memory, then the online algorithm requires $T = 2^{2(64-48)} = 2^{32}$ time. This computation can be divided across $T' = N/M = 2^{16}$ processors, each performing 2^{16} operations, and requiring (a single) access to $M' = M/T' = 2^{32}$ memory (i.e., 32 gigabytes).

4.2 Parallel Collision Search [24]

The parallel collision search algorithm was published by van Oorschot and Wiener [24], reducing the memory required for finding collisions in an n -bit function F (compared to trivial algorithms). Given $M = 2^m$ words of memory, the algorithm builds a chain structure which is similar to a Hellman table. The chain structure contains 2^m chains, where each chain starts at an arbitrary point and is terminated at a distinguished point (stored in memory) such that its average length is $T' = 2^{(n-m)/2}$ (i.e., the distinguished point set is of size N/T'). As in the case of a Hellman table, since $T' \cdot T'M = N$, then every chain is expected to collide with about one other chain in the structure. Thus, the structure contains about M' collisions which can be recovered efficiently as shown in [24]. However, as our attacks only use a degenerated variant of this algorithm, this short description suffices in order to understand the rest of this paper.

4.3 Basic Time-Memory-Data Tradeoff Attacks on the FX-Construction [18](without Preprocessing)

We describe the basic and well-known chosen plaintext attack of the FX-construction [18], which is based on the attack of Daemen on the Even-Manour

scheme [10]. We also mention the known plaintext attack with the same complexity on the scheme (see [12]), but we do not describe it here, as it is less relevant for this paper. On the other hand, the ideas and notation introduced in this simple attack will be repeatedly used throughout the rest of this paper.

The attack on the FX-construction is based on encrypting plaintexts P_i , and independently evaluating the core function F_K with values of K and X_j , while looking for an (i, j) pair such that $P_i \oplus K_1 = X_j$. In order to detect such a collision efficiently, we cancel the effect of the masking keys by defining functions $\phi_1(P_i)$ and $\phi_2(K, X_j)$ such that $P_i \oplus K_1 = X_j$ implies that $\phi_1(P_i) = \phi_2(K, X_j)$. These functions enable us to efficiently filter the (i, j) candidates.

For a general FX-construction, we pre-fix an arbitrary value $\Delta \neq 0$, set $P'_i \triangleq P_i \oplus \Delta$ and $X'_j \triangleq X_j \oplus \Delta$, and define:

$$\phi_1^{FX}(P_i) \triangleq C_i \oplus C'_i = FX_{K, K_1, K_2}(P_i) \oplus FX_{K, K_1, K_2}(P_i \oplus \Delta)$$

$$\phi_2^{FX}(K, X_j) \triangleq Y_j \oplus Y'_j = F_K(X_j) \oplus F_K(X_j \oplus \Delta).$$

Thus, $P_i \oplus K_1 = X_j$ implies that $\phi_1^{FX}(P_i) = FX_{K, K_1, K_2}(P_i) \oplus FX_{K, K_1, K_2}(P_i \oplus \Delta) = K_2 \oplus F_K(K_1 \oplus P_i) \oplus K_2 \oplus F_K(K_1 \oplus P_i \oplus \Delta) = F_K(X_j) \oplus F_K(X_j \oplus \Delta) = \phi_2^{FX}(K, X_j)$ as required. Note that each collision gives candidates for the full key ($K, K_1 = P_i \oplus X_j, K_2 = C_i \oplus Y_j$), which can be easily tested using trial encryptions.

For an SFX-construction in which $K_2 = A(K_1)$ (such as PRINCE and PRIDE), the functions $\phi_1(P_i)$ and $\phi_2(K, X_j)$ can be simplified to use single pairs of (P_i, C_i) and (X_j, Y_j) , respectively. Formally, we define:

$$\phi_1^{SFX}(P_i) \triangleq A(P_i) \oplus C_i = A(P_i) \oplus FX_{K, K_1, K_2}(P_i)$$

$$\phi_2^{SFX}(K, X_j) \triangleq A(X_j) \oplus Y_j = A(X_j) \oplus F_K(X_j).$$

Thus, $P_i \oplus K_1 = X_j$ implies that $\phi_1^{SFX}(P_i) = A(P_i) \oplus FX_{K, K_1, K_2}(P_i) = A(P_i) \oplus K_2 \oplus F_K(K_1 \oplus P_i) = A(P_i \oplus K_1) \oplus F_K(K_1 \oplus P_i) = A(X_j) \oplus F_K(X_j) = \phi_2^{SFX}(K, X_j)$ as required.

The details of the attack are given in Appendix A. It has a memory complexity of D and an expected time complexity of $\max(2D, 2^{\kappa+n-d+1})$ on FX-constructions. For SFX-constructions, the data and time complexities of the attack are reduced by a factor of 2.

4.4 Time-Memory-Data Tradeoff Attacks on Even-Mansour [14] (with Preprocessing)

We now summarize the time-memory-data tradeoff by Fouque et al. on the Even-Mansour scheme, which uses a one-time preprocessing phase. The main idea of the attack is to request plaintexts P_i , and evaluate the permutation F with values X_j such that a collision $P_i \oplus K_1 = X_j$ can be efficiently detected. In order to do so, we request the data P_i and evaluate values X_j in the form of chains, as in the parallel collision search algorithm [24].

Although we cannot immediately detect that $P_i \oplus K_1 = X_j$, the main observation of Fouque et al. is that we can independently add to P_i and X_j the same value $\phi_1(P_i) = \phi_2(X_j)$ (for the functions ϕ_1, ϕ_2 defined above for the FX and SFX constructions, where the key of the core cipher K is simply be ignored), which guarantees that in case $P_i \oplus K_1 = X_j$, then $P_{i+1} \oplus K_1 = X_{j+1}$.⁹ Hence, the functions we iterate are defined as

$$P_{i+1} \triangleq \Phi_1(P_i) \triangleq P_i \oplus \phi_1(P_i)$$

$$X_{j+1} \triangleq \Phi_2(X_j) \triangleq X_j \oplus \phi_2(-, X_j).$$

Note that Φ_1 and Φ_2 are generally non-bijective mappings (rather than permutations), and their behaviour (and in particular, the analysis of their collision probability) can be modeled using random functions, assuming that the underlying cipher does not behave unexpectedly.¹⁰

The downside of the approach of iterating the defined functions is that the attack becomes an adaptively-chosen plaintext attack, as $P_{i+1} = P_i \oplus \phi_1(P_i)$ depends on C_i and cannot be computed in advance (both for general FX-constructions and SFX-constructions).

The attack works by evaluating chains during the preprocessing phase, where each chain is iterated using Φ_2 and terminated at a distinguished point that is stored in memory. During the online phase, we evaluate a chain iterated using Φ_1 and terminated at a distinguished point. The online distinguished point is matched with the ones stored in memory, where a match allows to recover the key.

The full details of the attack are described in Appendix B. The time and data complexities of the online phase of the attack are both about $T = D = 2^d$ for SFX-constructions and 2^{d+1} for FX-constructions. Its memory complexity is $M = 2^{n-2d}$, and it requires preprocessing time of $\hat{T} = 2^{n-d}$ for SFX-constructions and $2N/D = 2^{n-d+1}$ for FX-constructions.

4.5 Time-Memory-Data Tradeoff attacks on the FX-Construction [14] (with Preprocessing)

As described in [14], we can easily generalize the previous attack on Even-Mansour to FX-constructions in which the internal permutation is keyed. We simply iterate over the 2^κ keys of the internal permutation, and preprocess each one separately by computing and storing its distinguished points. In total, we have $M = 2^\kappa \cdot 2^{n-2d} = 2^{\kappa+n-2d}$, while the preprocessing time is about $\hat{T} = 2^{\kappa+n-d}$ for SFX-constructions and $\hat{T} = 2^{\kappa+n-d+1}$ for FX-constructions. The online phase of the attack is essentially the same as in the previous attack, i.e., $T = D = 2^d$ for SFX-constructions and 2^{d+1} for FX-constructions.

⁹ The paper of [14] refers to this situation as the chains becoming parallel.

¹⁰ An exception is the specific case of SFX-constructions where the masking keys are equal (the affine mapping A is the identity), and hence Φ_1 and Φ_2 defined with the corresponding ϕ_1^{SFX} and ϕ_2^{SFX} are permutations. Thus, in this particular case, we use Φ_1 and Φ_2 defined with the more general ϕ_1^{FX} and ϕ_2^{FX} , resulting in slightly less efficient attacks.

5 New Time-Memory-Data Tradeoff Attacks on the FX-Construction without Preprocessing

In this section, we describe our new time-memory-data tradeoff attacks on the FX-construction, without using a preprocessing phase. The attacks are described in the most general form, i.e., they are applicable to general FX-constructions (exploiting the general definitions of ϕ_1, ϕ_2 , given in Section 4.3). However, as this paper focuses on the concrete SFX-constructions PRINCE and PRIDE, we directly analyze only the variants of the attacks which are optimized for SFX-constructions (i.e., assuming $\phi_1 = \phi_1^{SFX}, \phi_2 = \phi_2^{SFX}$). In order to calculate the complexity parameters for general FX-constructions, we simply multiply the data and time complexities of the attack by a factor of 2, as in the attacks of sections 4.3, 4.4 and 4.5.

5.1 The Case of $D \leq 2^{n/2}$

The attack for the case of $D \leq 2^{n/2}$ can be considered as a straightforward extension of the attack of [14] on the FX-construction (described in Section 4.5). However, [14] focused on attacks with preprocessing on the FX-Construction, and attacks without preprocessing were not described.

We extend the iteration function Φ_2 (defined in the Even-Mansour attack of the FX-construction in Section 4.4) by adding the key of the core cipher as a parameter. The iteration functions are now defined¹¹ as

$$P_{i+1} \triangleq \Phi_1(P_i) \triangleq P_i \oplus \phi_1(P_i)$$

$$X_{j+1} \triangleq \Phi_2(K, X_j) \triangleq X_j \oplus \phi_2(K, X_j).$$

The attack is described below:

1. Build a chain of plaintexts, starting from an arbitrary plaintext, extended using the iteration function $P_{i+1} = \Phi_1(P_i)$, and terminated at a distinguished point \hat{P} for which the $\log(D)$ LSBs of $\phi_1(\hat{P})$ are 0 (as in the attack of Appendix B). Store the endpoint \hat{P} , and its value $\phi_1(\hat{P})$.
2. For each possible value of K :
 - (a) For N/D^2 different starting points X_0 :
 - i. Build a chain starting from X_0 , defined according to $X_{j+1} = \Phi_2(K, X_j)$, and terminated at a distinguished point \hat{X} for which the $\log(D)$ LSBs of $\phi_2(K, \hat{X})$ are 0. If $\phi_2(K, \hat{X}) = \phi_1(\hat{P})$, test the full key K, K_1, K_2 derived from \hat{P} and \hat{X} using a trial encryption.

¹¹ Where ϕ_1 and ϕ_2 are defined in Section 4.3.

For the correct value of K in Step 2, we expect a collision between a node in the online chain (of average length D) and the (expected number of) N/D nodes evaluated offline. As this collision causes the corresponding chains to merge, it will be detected at the next distinguished point, allowing to recover the key.

The expected data complexity of the attack is $D = 2^d$, while its memory complexity is negligible. The total number of distinguished points that we compute in Step 2 is about $2^{\kappa+n-2d}$, requiring about $2^{\kappa+n-2d+d} = 2^{\kappa+n-d}$ computation time. For each such distinguished point, we do not perform more than one trial encryption, and therefore the expected total time complexity of the attack is $2^{\kappa+n-d}$.

Consequently, we obtain about the same data and time complexities as the attack described in Section 4.3, but (almost) completely nullify the memory complexity in the adaptively chosen plaintext model.

5.2 The Case of $D > 2^{n/2}$

The attack for $D \leq 2^{n/2}$ has to be adapted for the case of $D > 2^{n/2}$, as the online chain (built in Step 1 of the previous attack) is expected to cycle (i.e., collide with itself) after about $2^{n/2}$ evaluations, and thus cannot cover more than $2^{n/2}$ nodes. Therefore, we build the structure of chains online, while evaluating offline only one chain per key.

We note that while this attack can also be viewed as an extension of the attacks of [14], it is less straightforward, as all the attacks of [14] use $D \leq 2^{n/2}$. The reason is that the attacks of [14] are based on the basic Even-Mansour attack (described in Section 4.4), for which $\kappa = 0$ and there is no gain in using $D > 2^{n/2}$ (as this increases the total time complexity of the attack). On the other hand, we observe that for FX-constructions, we can indeed benefit from $D > 2^{n/2}$.

1. For D^2/N different starting points P_0 :
 - (a) Build a chain of plaintexts, starting from P_0 , extended using the formula $P_{i+1} = \Phi_1(P_i)$, and terminated at a distinguished point \hat{P} for which the $\log(N/D)$ LSBs of $\phi_1(\hat{P})$ are 0. Store the endpoint \hat{P} in a list L , sorted according to $\phi_1(\hat{P})$.
2. For each possible value of K :
 - (a) Build a chain starting from an arbitrary value X_0 , defined according to $X_{j+1} = \Phi_2(K, X_j)$, and terminated at a distinguished point \hat{X} for which the $\log(N/D)$ LSBs of $\phi_2(K, \hat{X})$ are 0. Search for $\phi_2(K, \hat{X})$ in the list L and for each match with some $\phi_1(\hat{P})$, recover \hat{P} , obtain a suggestion for the full key K, K_1, K_2 and test it using a trial encryption.

The chain structure of plaintexts covers $D^2/N \cdot N/D = D$ nodes on average, and it is expected to collide with the chain of values (of expected length N/D)

for the correct K , allowing to recover it. The data complexity of the attack is $D = 2^d$, while its memory complexity is $M = D^2/N = 2^{2d-n}$.

Computing the distinguished points online and offline requires $\max(2^d, 2^{\kappa+n-d}) = 2^{\kappa+n-d}$ time (assuming $\kappa \geq n$). Two arbitrary distinguished points match with probability $2^{(n-d)-n} = 2^{-d}$ (as the $n-d$ LSBs of distinguished points always match). We store a total of 2^{2d-n} distinguished points in L , and evaluate a total of 2^κ distinguished points in Step 2. Thus, the expected number of matches (resulting in trial encryptions) is $2^{(\kappa+2d-n)-d} = 2^{\kappa-n+d} \leq 2^{\kappa+n-d}$ (as $d \leq n$), and the expected total time complexity is $2^{\kappa+n-d}$, dominated by the computation of the distinguished points in Step 2.

Consequently, we obtain (about) the same time complexity as the basic attack of Section 4.3, but gain a factor of $2^d/(2^{2d-n}) = 2^{n-d}$ in memory.

6 New Time-Memory-Data Tradeoff Attacks on the FX-Construction with Preprocessing

In this section, we describe our new time-memory-data tradeoff attacks on the FX-construction, taking advantage of a preprocessing phase. As in Section 5, we directly analyze only the attack variants which are optimized for SFX-constructions (although the attacks are described in the most general form). For general FX-constructions, we simply multiply the data and time complexities of the attacks by a factor of 2.

6.1 The Case of $D \leq 2^{n/2}$

It is possible to apply standard time-memory-data tradeoffs for stream ciphers [3, 4] to the FX-construction in the chosen plaintext model (and to some extent, also in the known plaintext model). However, the most interesting tradeoffs are obtained in the adaptively chosen plaintext model, in which we combine the attacks of the previous section with techniques borrowed from stream cipher cryptanalysis.

We use Hellman's time-memory tradeoff algorithm in order to cover during the preprocessing phase of the attack, the $2^{\kappa+n-2d}$ pairs of (K ,distinguished point) that were computed in Step 2 of the attack of Section 5.1. These pairs were all stored in memory in the attack of [14] (described in Section 4.5), which required (at least) 2^κ words of storage. This memory complexity is completely impractical for standard values of $\kappa \geq 64$, and our techniques trade it off with the online time complexity.

The idea of using Hellman's time-memory tradeoff algorithm to cover special points was first published in cryptanalysis of a certain type of stream ciphers,¹² and we now show how to adapt it to the FX-construction. In order to cover the pairs of (K ,distinguished point), we define a mapping between the $2^{\kappa+n-2d}$

¹² Refer to tradeoffs for stream ciphers with low sampling resistance [3, 4].

pairs, denoted by $h_2(K, \hat{X})$. One problem that we need to overcome is that \hat{X} is an n -bit word, and does not contain the sufficient $\kappa + n - 2d \geq n$ bits¹³ in order to define this mapping. Furthermore, the mapping cannot directly depend on K , as in the online phase we search the Hellman tables without knowledge of the online key. Thus, in order to collect more data, we simply continue evaluating the chain by applying Φ_2 to (K, \hat{X}) sufficiently many times, until we collect the $\kappa + n - 2d$ bits required in order to define the *Hellman value* of (K, \hat{X}) . This value will be used in order to determine the next $(K, \text{distinguished point})$ pair, i.e., the output of $h_2(K, \hat{X})$. The algorithm of the Hellman mapping $h_2(K, \hat{X})$ is given below, assuming that $\kappa = n$ for the sake of simplicity. We note that in cases where $\kappa > n$, we simply apply Φ_2 more times in order to collect more data (the case of $\kappa < n$ can be handled by truncation).

1. Compute the $2n$ -bit Hellman value of (K, \hat{X}) by first computing the next 2 points in the chain $X' = \Phi_2(K, \hat{X})$ and $X'' = \Phi_2(K, X')$. The Hellman value of (K, \hat{X}) is defined as $(\phi_2(K, X'), \phi_2(K, X''))$.
2. Interpret the Hellman value as $(Z, K^{next}) = (\phi_2(K, X'), \phi_2(K, X''))$ (note that both Z and K^{next} are n -bit words). Compute a chain of (average) length $D = 2^d$, using the iteration function $\Phi_2(K^{next}, X)$, starting from $X = Z$, and terminating at a distinguished point $(K^{next}, \hat{Z}^{next})$ (i.e., the $\log(D)$ LSBs of $\phi_2(K^{next}, \hat{Z}^{next})$ are zero). Output $h_2(K, \hat{X}) = (K^{next}, \hat{Z}^{next})$.

Once the mapping $h_2(K, \hat{X})$ is well-defined, we can use Hellman's preprocessing algorithm to cover a space of $2^{\kappa+n-2d}$ points (pairs) (K, \hat{X}) . As the average time complexity of one application of $h_2(K, \hat{X})$ is D , the total time complexity of the preprocessing phase is $\hat{T} = D \cdot 2^{\kappa+n}/D^2 = 2^{\kappa+n-d}$. Since h_2 is defined on (at most) $2n$ bits, we can store $M/2$ chains with M words of memory. However, in case $D \approx 2^{n/2}$ and $\kappa \leq n$, we essentially need to cover a space of at most 2^n (we cover one distinguished point per key on average), and thus we can store a larger number of M chains with M words of memory.

We now point out a few technical issues about the preprocessing algorithm: since we are using Hellman's algorithm to cover $2^{\kappa+n-2d}$ points, the (average) length of the Hellman chains is $2^{\kappa+n-2d-m}$ (determined according to the available memory $M = 2^m$). In order to terminate a Hellman chain (computed using h_2 on the space of $\kappa + n - 2d$ bits), we need to define a subset of "Hellman distinguished points", containing pairs of (K, \hat{X}) . Such a subset (which determines when to terminate an iteration chain of h_2) can be defined (for example) according to the LSBs of the Hellman value $(\phi_2(K, X'), \phi_2(K, X''))$, computed in Step 1 of the algorithm above. The "Hellman distinguished points" should be contrasted with the distinguished points defined for the iteration on the n -bit

¹³ We consider $\kappa = n$ and $d \leq n/2$, and thus $\kappa + n - 2d \geq n$.

space with a fixed key using Φ_2 (such distinguished points are defined according to the LSBs of $\phi_2(K, \hat{X})$). In order to avoid confusion, we refer to chains and distinguished points computed using h_2 as *Hellman chains* and *Hellman distinguished points*, whereas the ones computed using Φ_1 and Φ_2 are simply referred to as (standard) chains and distinguished points.¹⁴ An additional technical issue is that in order to cover the full space of $2^{\kappa+n-2d}$ points, we need to define flavors of h_2 (namely, $h_2^{[i]}$), and this can be done (for example) by defining $h_2^{[i]} = h_2(K, \hat{X}) + (i, i)$.

The online algorithm is given below.

1. Compute a chain of (approximately) D points using the iteration function ϕ_1 , starting from an arbitrary plaintext, and terminating at a distinguished point \hat{P} , where the $\log(D)$ LSBs of $\phi_1(\hat{P})$ are 0.
2. Given \hat{P} , compute the corresponding Hellman value $(\phi_1(P'), \phi_1(P''))$ similarly to the preprocessing phase, by computing the next 2 points in the chain $P' = \Phi_1(\hat{P})$ and $P'' = \Phi_1(P')$.
3. Invert $(K', X') = (\phi_1(P'), \phi_1(P''))$ using the Hellman tables, obtain a suggestion for the full key (K, K_1, K_2) and test it.

The data complexity of the attack is D , and according to Hellman's time-memory tradeoff curve, its average time complexity is $T' = (N'/M')^2$ evaluations of h_2 , where $N' = 2^{\kappa+n-2d}$ is the size of the covered space and $M' = M/2 = 2^{m-1}$ is the number of Hellman chains stored in memory. Thus $T' = (2^{\kappa+n-2d}/2^{m-1})^2 = 2^{2(\kappa+n-m-2d+1)}$, and since each evaluation of h_2 requires $D = 2^d$ time, then $T = 2^{2(\kappa+n-m-1.5d+1)}$. When $D \approx 2^{n/2}$ and $\kappa \leq n$, we can use the M memory words more efficiently and obtain a (slightly) improved time complexity of $T = 2^{2(\kappa+n-m-1.5d)}$.

The Difference Between Tradeoffs for FX-Constructions and Stream Ciphers. As can be seen from the tradeoff above for FX-constructions, the effective key size is reduced by a factor of $2^{1.5d}$ when obtaining 2^d data. On the other hand, for stream ciphers, the effective hidden state size is only reduced by a factor of 2^d . The reason for this is that in the case of stream ciphers, the state update function is typically a permutation.¹⁵ On the other hand, the corresponding function in the case of FX-constructions is Φ_1 , which is a non-bijective function rather than a permutation. Iterating Φ_1 a large number of times results in entropy loss due to collisions in its functional graph. This reduces the number of points that we need to search in the Hellman tables to recover

¹⁴ Our definitions are related to the definitions of *full name*, *output name*, and *short name* in the context of stream ciphers with low sampling resistance [3].

¹⁵ For example, many stream ciphers are built using feedback shift registers, and it is possible to run them backwards in a deterministic way.

the key, and improves the complexity of the online attack compared to the case of stream ciphers.

Implementation for $n = \kappa = 64$. In the case of PRINCE and PRIDE, then $n = \kappa = 64$. We assume that we have $2^m = 2^{48}$ words of memory (2^{51} bytes) and we can obtain $2^d = 2^{32}$ adaptively chosen plaintexts. In total, the online time complexity of the algorithm is $2^{2(64+64-48-48)} = 2^{64}$, corresponding to Attack 1 in Table 1. Similarly to the example given at the end of Section 4.1, as the Hellman chains cover a space of size $2^\kappa = 2^{64}$ points, the 2^{64} computation can be divided across 2^{16} CPUs, each requiring (a single) access to a memory of 2^{32} words (and it can thus be stored on a hard disk). Each CPU invokes h_2 about 2^{16} times, where each invocation requires 2^{32} cipher evaluations. Thus, each CPU performs about 2^{48} cipher evaluations in total.

6.2 The Case of $D > 2^{n/2}$

As in the case considered in Section 5.2, we need to adapt the previous attack to efficiently exploit $D > 2^{n/2}$ data. Similarly to the case of $D \leq 2^{n/2}$, the techniques we use for $D > 2^{n/2}$ are related to those of [3, 4]. However, as we describe next, the method in which we request the data and optimize the parameters in this setting are different from the stream cipher setting (where the method in which the keystream is obtained does not seem to influence the complexity of the attack).

We consider two different adaptation methods to the previous attack of $D \leq 2^{n/2}$. In the first method, we obtain the data using chains of (maximal) length $2^{n/2}$. In order to ensure that these chains do not merge, we define flavors of Φ_1 and Φ_2 (which should be contrasted with the Hellman flavors of h_2). Thus, we define $2^{d-n/2}$ flavors, and build Hellman tables for each one. During the online phase, we obtain one distinguished point per flavor and search it in the corresponding Hellman tables. One can observe that in terms of the time-memory tradeoff, the flavors of Φ_1 and Φ_2 play a similar role to the flavors of h_2 in the attack with $d = n/2$. Consequently, we obtain the same time-memory tradeoff as for $d = n/2$, i.e., $T = 2^{2(\kappa+n/4-m)}$.¹⁶ On the other hand, the preprocessing complexity is reduced to $\hat{T} = 2^{\kappa+n-d}$.

An Improved Tradeoff. The attack above is a direct extension of the tradeoff obtained for $D = 2^{n/2}$, which covered offline, 2^κ distinguished points using h_2 . We now show how to obtain an improved attack, using a simple and yet subtle and non-trivial observation. We notice that for $D > 2^{n/2}$, we can use chains of length $N/D < 2^{n/2}$. These chains are shorter than the ones used for $D = 2^{n/2}$ data, and are of the same length as in the attack with only $2^{d'} = 2^{n-d}$ data. At first, it may not be clear why using shorter chains results in a better attack. As we show next, the reason for this is that we can use the memory more efficiently than in the case of longer chains.

¹⁶ There are additional restrictions to this curve, discussed at the end of the section.

We first compare our case of $2^d > 2^{n/2}$ data and the case of $2^{d'} = 2^{n-d}$ data, which use the same chain length. The difference is that in the case of $2^{d'} = 2^{n-d}$ data, we obtained only one distinguished point online, whereas now we have $2^{d-(n-d)} = 2^{2d-n}$ such distinguished points, and we need to cover only one of them offline in order to succeed. Thus, in the attack with $2^{d'} = 2^{n-d}$ data, we had to cover offline, the large space of $2^{\kappa+n-2d'}$ distinguished points using Hellman tables, whereas now we need to cover only $2^{\kappa+n-2d'-(2d-n)} = 2^{\kappa+n-2(n-d)-(2d-n)} = 2^\kappa$ distinguished points. Namely, we need to cover about one distinguished point for every key K of the core cipher, as in the attack above. However, the crucial observation is that the space of distinguished points is of the same size as in the attack with $2^{d'} = 2^{n-d}$ data. This space is larger than the space for $D = 2^{n/2}$, implying that we can build larger Hellman tables and use the memory more efficiently compared to the (non optimal) attack above (which is a direct extension of the case of $D = 2^{n/2}$).

A simple way to compute the improved tradeoff is to start with the formula $T = 2^{2(\kappa+n'-m-1.5d')}$, calculated for the attack with $d' \leq n/2$. Then, we plug in $d' = n - d$ and $n' = n - (2d - n)$, as the space size that we cover by Hellman tables is reduced by a factor of 2^{2d-n} (which is the number of distinguished points obtained online). In other words, the tradeoff $T = 2^{\kappa+n-(2d-n)-m-1.5(n-d)} = 2^{2(\kappa+n/2-m-d/2)}$ is obtained by reducing the number of Hellman tables (by a factor of 2^{2d-n}) compared to the attack that used $d' = n - d$. However, the attack cannot use less than 1 Hellman table, and it is therefore necessary to derive an expression for this variable, which restricts the tradeoff. Interestingly, the simplest method that we found to compute the number of Hellman tables is to redo the low-level computation, which also gives a better understanding of the full attack.

For parameters T' and M' , we build a Hellman table of distinguished points (using the function h_2) with the stopping rule of

$$T' \cdot T' M' = 2^{\kappa+n-2d'} = 2^{\kappa-n+2d} \quad (1)$$

(after which the Hellman chains start colliding extensively).¹⁷ We need to cover about 2^κ distinguished points with H Hellman tables, namely

$$HT' M' = 2^\kappa. \quad (2)$$

Since each evaluation of h_2 requires 2^{n-d} time, the preprocessing time complexity is $\hat{T} = 2^{\kappa+n-d}$ (as in the non-optimized attack above). The total memory complexity of the attack is

$$M = HM' \quad (3)$$

and the total online time complexity is calculated as follows: searching a single Hellman table requires T' evaluations of h_2 , i.e., a total of $T' \cdot 2^{n-d}$ time. For each of the 2^{2d-n} distinguished points, we need to search the H Hellman tables, and thus the total online time complexity is

$$T = T' \cdot 2^{n-d} \cdot H \cdot 2^{2d-n} = T' H \cdot 2^d. \quad (4)$$

¹⁷ Note that the stopping rule in the previous attack was $T' \cdot T' M' = 2^\kappa < 2^{\kappa-n+2d}$.

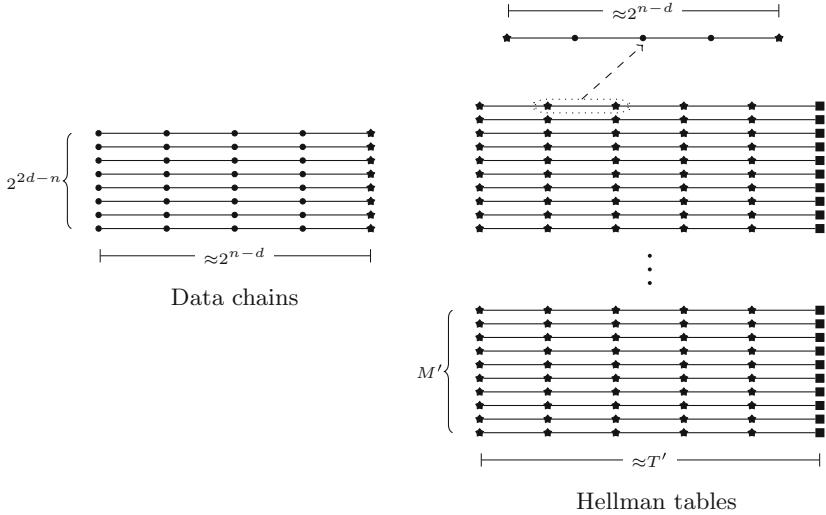


Fig. 2. Time-Memory-Data Tradeoff with Preprocessing for $D > 2^{n/2}$

We calculate the tradeoff according to (3) and (4) by evaluating $T \cdot M^2 = T' H \cdot 2^d \cdot (HM')^2 = 2^d \cdot H^3 \cdot T' \cdot (M')^2$. From (2), we get $T \cdot M^2 = 2^{\kappa+d} \cdot H^2 \cdot M'$. Furthermore, from (1) and (2), we obtain

$$H^2 \cdot M' = 2^{2\kappa - (\kappa - n + 2d)} = 2^{\kappa + n - 2d}. \quad (5)$$

Thus, $T \cdot M^2 = 2^{\kappa+d+\kappa+n-2d} = 2^{2\kappa+n-d}$, i.e., we obtain the tradeoff

$$T = 2^{2(\kappa+n/2-m-d/2)}$$

(which was obtained above in a different way). This tradeoff efficiently exploits more than $2^{n/2}$ data, unlike the previous tradeoff $T = 2^{2(\kappa+n/4-m)}$.

As noted above, a condition that we have to impose on this tradeoff is that the number of Hellman tables is at least 1, i.e., $H \geq 1$. In order to calculate H , we use (3) and (5), obtaining $H = 2^{\kappa+n-2d-m}$. Since $H \geq 1$, the tradeoff above is valid only for $m \leq \kappa + n - 2d$.

When we want to utilize 2^m memory for $m > \kappa + n - 2d$, then we use only one Hellman table (i.e., $H = 1$), and we are forced to stop the Hellman chains before the stopping rule $(T')^2 \cdot M' < 2^{\kappa-n+2d}$ (1). Namely, we have $M' = M = 2^m$ and $T' M' = 2^\kappa$, implying that $T' = 2^{\kappa-m}$, and using (4), $T = T' \cdot 2^d = 2^{\kappa+d-m}$. Note that $(T')^2 \cdot M' = 2^{2(\kappa-m)} \cdot 2^m = 2^{2\kappa-m} < 2^{2\kappa-(\kappa+n-2d)} = 2^{\kappa-n+2d}$, so indeed we do not violate the stopping rule (1).

Finally, we observe that a similar restriction on m also applies to the previous tradeoff $T = 2^{2(\kappa+n/4-m)}$ for $d > n/2$, and it is possible to show that the tradeoff obtained here is always at least as efficient as the previous one.

Implementation for $n = \kappa = 64$. We assume that we have $2^m = 2^{48}$ words of memory and we can obtain $2^d = 2^{40}$ adaptively chosen plaintexts. In

total, the online time complexity of the algorithm is $T = 2^{2(\kappa+n/2-m-d/2)} = 2^{2(64+32-48-20)} = 2^{56}$, corresponding to Attack 2 in Table 1. In this case $H = 2^{\kappa+n-2d-m} = 1$, i.e., we have a single Hellman table. As we search the table with $2^{2d-n} = 2^{16}$ distinguished points, the 2^{56} computations can be divided across (up to) 2^{16} CPUs, each performing $2^{56-16} = 2^{40}$ computations, and accessing the memory only once (and it can therefore be stored on a hard disk).

7 Conclusions

In this paper, we proposed new generic time-memory-data tradeoffs for FX-constructions, and optimized them for the recent proposals PRINCE and PRIDE. Some of our attacks are surprisingly efficient, and despite their limitations, we believe that they demonstrate the small security margin of PRINCE and PRIDE against practical attacks. In the extended version of this paper [11], we show that PRINCE and PRIDE could counter these generic attacks with little overhead by incorporating the masking keys into the key schedule of the core ciphers. This suggests that the DESX solution proposed by Ron Rivest in 1984 (in order to provide better security for the widely-deployed DES) may be less suitable for new ciphers.

Acknowledgments. The author would like to thank Orr Dunkelman and Adi Shamir for helpful discussions on this work.

A Details of the Basic Time-Memory-Data Tradeoff Attack on the FX-Construction [18] (without Preprocessing)

We give the details of the basic attack using the general functions $\phi_1(P_i)$ and $\phi_2(K, X_j)$, defined in Section 4.3.

1. Obtain the encryptions of D arbitrary values P_i , denoted by C_i . For general FX-constructions, also obtain the D additional encryptions required to calculate $\phi_1^{FX}(P_i)$, by asking for the encryptions of $P'_i = P_i \oplus \Delta$. Store (P_i, C_i) in a list L , sorted according to $\phi_1(P_i)$.
2. For each possible value of K :
 - (a) For N/D different values of X_j :
 - i. Compute $\phi_2(K, X_j)$ by computing $Y_j = F_K(X_j)$ (for a general FX-construction, also compute $F_K(X_j \oplus \Delta)$).
 - ii. Search $\phi_2(K, X_j)$ in L . For each match, retrieve (P_i, C_i) , and test each of the key candidate triplet $(K, K_1 = P_i \oplus X_j, K_2 = C_i \oplus Y_j)$ using trial encryptions. If a trial encryption succeeds, return the corresponding full key (otherwise return to Step 2.(a)).

According to the birthday paradox, for the correct value of K in Step 2, we expect a pair (i, j) such that $P_i \oplus K_1 = X_j$. Therefore, we expect to obtain a match in L in Step 2.(a).ii between $\phi_1(P_i)$ and $\phi_2(K, X_j)$ and recover the correct K, K_1, K_2 .

For general FX-constructions, the data complexity of the attack is $2D$ chosen plaintexts, and its memory complexity is $M = 2D$ n -bit words, required in order to store L . In order to compute the time complexity, we note that for an arbitrary value of the n bits of $\phi_2(K, X_j)$, we expect at most one match in L (which contains at most 2^n elements). Thus, the expected time complexity of Step 2.(a) is about 2, implying that the expected time complexity of the full attack is $\max(2D, 2^{\kappa+n-d+1})$ (i.e., we can efficiently exploit $D \leq 2^{(\kappa+n)/2}$ data). For SFX-constructions, the data and time complexities of the attack are reduced by a factor of 2 (note that in this case, the attack requires only known plaintexts).

B Details of the Time-Memory-Data Tradeoff Attack on Even-Mansour [14] (with Preprocessing)

We assume that we can obtain the encryptions of about $D \leq 2^{n/2}$ adaptively-chosen plaintexts during the online phase. During the preprocessing phase, we use the preprocessing iteration function $\Phi_2(X)$ (defined in Section 4.4) in order to build a structure containing N/D^2 chains. Each chain is evaluated from an arbitrary starting point, and terminated at a distinguished point \hat{X} for which the $\log(D)$ LSBs of $\phi_2(\hat{X})$ are zero. Thus, the average chain length is D , implying that the time complexity of preprocessing is $\bar{T} = N/D = 2^{n-d}$ for SFX-constructions and $2N/D = 2^{n-d+1}$ for FX-constructions. For each chain in the structure, we store in memory only the endpoint¹⁸ \hat{X} , and sort the chains according to their values $\phi_2(\hat{X})$. Thus, the memory complexity is about $M = N/D^2 = 2^{n-2d}$.

During the online phase, we evaluate a single chain of (expected) length D , starting for an arbitrary plaintext. The chain is defined according to the online iteration function $\Phi_1(P)$, and is terminated at a distinguished point \hat{P} for which the $\log(D)$ LSBs of $\phi_1(\hat{P})$ are zero. Once a distinguished point is reached, we search for it in the structure, and for each match, we obtain and test the key suggestions for K_1, K_2 . Note that unlike Hellman's original attack, we directly recover the key from the distinguished points stored in the structure, without the need to further traverse the chains (and thus we do not need to store any information about their startpoints).

As the offline structure covers about 2^{n-d} values of X_j and the online chain contains 2^d values of P_i , we expect a collision $P_i \oplus K_1 = X_j$. The collision implies that $\phi_1(P_i) = \phi_2(X_j)$, which causes the two corresponding chains to merge and reach distinguished points with the same value. This distinguished

¹⁸ The structure is somewhat different from a Hellman table, for which we also store information about the startpoints of the chains.

point is recovered in the online phase and allows to recover the key K_1, K_2 . Thus, the time and data complexities of the online phase of the attack are both about $T = D = 2^d$ for SFX-constructions and 2^{d+1} for FX-constructions.

References

1. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçın, T.: Block Ciphers – Focus on the Linear Layer (feat. PRIDE). In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 57–76. Springer, Heidelberg (2014)
2. Barkan, E., Biham, E., Shamir, A.: Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 1–21. Springer, Heidelberg (2006)
3. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)
4. Biryukov, A., Shamir, A., Wagner, D.: Real Time Cryptanalysis of A5/1 on a PC. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 1–18. Springer, Heidelberg (2001)
5. Biryukov, A., Wagner, D.: Advanced Slide Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 589–606. Springer, Heidelberg (2000)
6. Bitcoin network graphs. <http://bitcoin.sipa.be/>
7. Borghoff, J., et al.: PRINCE – A Low-latency Block Cipher for Pervasive Computing Applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
8. Borst, J., Preneel, B., Vandewalle, J.: On the Time-memory Tradeoff Between Exhaustive Key Search and Table Precomputation. In: Proceedings of 19th Symposium in Information Theory in the Benelux, WIC, pp. 111–118 (1998)
9. COPACOBANA faqs. <http://www.copacobana.org/faq.html>
10. Daemen, J.: Limitations of the Even-mansour Construction. In: Imai et al. (eds.) [17], pp. 495–498
11. Dinur, I.: Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE. Cryptology ePrint Archive, Report 2014/656 (2014). <http://eprint.iacr.org/>
12. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 336–354. Springer, Heidelberg (2012)
13. Even, S., Mansour, Y.: A Construction of a Cioher From a Single Pseudorandom Permutation. In: Imai et al. (eds.) [17], pp. 210–224
14. Fouque, P.-A., Joux, A., Mavromati, C.: Multi-user Collisions: Applications to Discrete Logarithm, Even-Mansour and PRINCE. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 420–438. Springer, Heidelberg (2014)
15. Güneysu, T., Kasper, T., Novotný, M., Paar, C., Rupp, A.: Cryptanalysis with COPACOBANA. IEEE Trans. Computers **57**(11), 1498–1513 (2008)
16. Hellman, M.E.: A Cryptanalytic Time-Memory Trade-Off. IEEE Transactions on Information Theory **26**(4), 401–406 (1980)
17. Imai, H., Rivest, R.L., Matsumoto, T. (eds.): ASIACRYPT 1991. LNCS, vol. 739. Springer, Heidelberg (1993)

18. Kilian, J., Rogaway, P.: How to Protect DES Against Exhaustive Key Search. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 252–267. Springer, Heidelberg (1996)
19. Mentens, N., Batina, L., Preneel, B., Verbauwheide, I.: Time-Memory Trade-Off Attack on FPGA Platforms: UNIX Password Cracking. In: Bertels, K., Cardoso, J.M.P., Vassiliadis, S. (eds.) ARC 2006. LNCS, vol. 3985, pp. 323–334. Springer, Heidelberg (2006)
20. National Institute of Standards and Technology. Recommendation for Key Management - Part 1: General (revision 3). NIST Special Publication 800-57 (2012)
21. Rivest, R.L.: DESX (1984) (never published)
22. Standaert, F.-X., Rouvroy, G., Quisquater, J.-J., Legat, J.-D.: A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results. In: Kaliski, B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 593–609. Springer, Heidelberg (2002)
23. The PRINCE Team. The PRINCE Challenge (2014). https://www.emsec.rub.de/research/research_startseite/prince-challenge/
24. van Oorschot, P.C., Wiener, M.J.: Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology* **12**(1), 1–28 (1999)

A Generic Approach to Invariant Subspace Attacks: Cryptanalysis of Robin, iSCREAM and Zorro

Gregor Leander^{1(✉)}, Brice Minaud², and Sondre Rønjom³

¹ Horst Görtz University for IT Security, Ruhr-Universität Bochum,
Bochum, Germany

gregor.leander@rub.de

² Agence Nationale de la Sécurité des Systèmes d'Information, Paris, France
brice.minaud@gmail.com

³ Nasjonal sikkerhetsmyndighet, Oslo, Norway
sondrer@gmail.com

Abstract. Invariant subspace attacks were introduced at CRYPTO 2011 to cryptanalyze PRINTCIPHER. The invariant subspaces for PRINTCIPHER were discovered in an ad hoc fashion, leaving a generic technique to discover invariant subspaces in other ciphers as an open problem. Here, based on a rather simple observation, we introduce a generic algorithm to detect invariant subspaces. We apply this algorithm to the CAESAR candidate iSCREAM, the closely related LS-design Robin, as well as the lightweight cipher Zorro. For all three candidates invariant subspaces were detected, and result in practical breaks of the ciphers. A closer analysis of independent interest reveals that these invariant subspaces are underpinned by a new type of self-similarity property. For all ciphers, our strongest attack shows the existence of a weak key set of density 2^{-32} . These weak keys lead to a simple property on the plaintexts going through the whole encryption process with probability one. All our attacks have been practically verified on reference implementations of the ciphers.

Keywords: Cryptanalysis · Lightweight cryptography · Invariant subspace · Self-similarity · iSCREAM · LS-designs · Zorro · CAESAR

Introduction

Block ciphers are one of the most essential cryptographic primitives. Our understanding of how to build secure block ciphers has greatly advanced in the last 20 years. Nowadays, analyzing a given block cipher with respect to a large class of non-trivial attacks, including linear and differential attacks and their variants, is a well-understood process for a large class of block ciphers. However,

G. Leander—The work of Gregor Leander was funded by the BMBF UNIKOPS project.

when it comes to designing block ciphers with strong performance requirements, often less conservative approaches are chosen. Examples of such performance requirements that have recently been studied extensively include low hardware footprint (e.g. PRESENT [6], LED [20], KATAN [10]), low memory consumption on small embedded processors (e.g. ITUBee [21], SPECK [5], PRIDE [2]), low latency (e.g. PRINCE [7]) and ease of side-channel protection (e.g. Zorro [14], LS-Designs [15]).

In order to fit within constrained settings, many of these ciphers feature innovative designs: they may rely on simpler round functions, or minimal key schedules. While in most cases, guarantees against traditional linear or differential attacks are still offered, the simpler structure of many of these ciphers may lend itself to new attacks. Careful cryptanalysis is required in order to assess the security of these new designs; in this process, new techniques have emerged.

One such technique is the invariant subspace attack, introduced in [22] for the cryptanalysis of PRINTCIPHER. The general idea behind this attack is the following: assume that the round function of a cipher maps a coset A of some vector subspace of the inner state to a coset B of the same space, and a fixed key belonging to $A - B$ is added in every round. Then the set A is preserved by the round function, and hence remains stable through the whole encryption process. This property holds for a large set of keys in PRINTCIPHER, breaking the cipher in a practical setting. This type of attack seems particularly well-suited to substitution-permutation networks (SPN) with a minimal key schedule or cryptographic permutations with highly structured round constants.

Invariant subspace attacks are unusual in that they rely on an unexpected form of symmetry in the round function, and yield attacks that are independent of the number of rounds. On the other hand, the same attributes are shared by attacks based on self-similarity properties. These properties were first formally defined in [4] to study alternative descriptions of AES, and later used in [8] to cryptanalyze the SHA-3 candidate *Lesamnta* and the lightweight cipher XTEA.

Interestingly, despite its fundamental nature, the understanding of symmetries and invariant subspaces, in block ciphers or cryptographic permutations, is rather limited. The invariant subspace attack on PRINTCIPHER was found in an ad hoc fashion and no general approach to detect or avoid such invariant spaces is known. This is even more surprising as for more involved attacks like differential and linear attacks and their variations our general understanding of detection and avoiding those attacks by design is much more evolved.

Our Contribution. In this paper we aim at increasing the general understanding of invariant subspaces. For this purpose, and as our first main result, we present a generic algorithm that is able to detect invariant subspaces. The running time of this algorithm depends on the block size of the primitive and the density of the existing invariant subspaces. In particular, it is especially efficient if relatively large invariant subspaces exist. As the impact of an invariant subspace increases with its dimension, this can be seen as detecting stronger attacks significantly faster than minor attacks.

We apply this generic algorithm to the lightweight cipher Robin introduced at FSE 2014 as a concrete instance of the LS-design framework [15], the closely related CAESAR [1] candidate iSCREAM [18], as well the lightweight cipher Zorro presented at CHES 2013 [14]. In all cases the algorithm is able to detect invariant subspaces.¹ Attacks resulting from these invariant subspaces break all three ciphers in a practical setting. All attacks have been verified on reference implementations of the ciphers.

As our second main contribution, we show that the invariant subspaces we have discovered are underpinned by a type of self-similarity property of independent interest, stemming from a linear map commuting with the round function. Surprisingly, such a map exists for all three ciphers, despite Robin and Zorro having quite different structures. As a result, we obtain stronger attacks on our target ciphers. We also hope to provide useful insight for the design and analysis of ciphers with minimal key schedules as well as for the choice of round constants in cryptographic permutations.

More specifically, our attacks show the existence of weak keys in Robin, Zorro, as well as iSCREAM in the chosen-tweak scenario. In all cases, the proportion of weak keys is 2^{-32} within the set of all keys. Encryption of a single chosen plaintext is enough to determine whether a key is weak, making our weak key setting very practical. Once a key is recognized as weak, a simple property on plaintexts goes through the whole encryption process with probability one, breaking plaintext confidentiality. In addition, the full 128-bit encryption key can be recovered using one chosen plaintext in time complexity 2^{64} . We also show that all three ciphers are instantly broken in the related-key setting, without any weak key requirement; although only iSCREAM claims security in this model.

In the case of LS-designs, we furthermore present a second attack, based on S-box-dependent invariant subspaces, without an underlying self-similarity. We obtain a new set of weak keys with the same properties as above, including the fact that they can be detected using a single chosen plaintext. However when applying this second attack to Robin and iSCREAM, we obtain a much rarer set of weak keys, with only one key in 2^{80} being weak. We then fine-tune this attack against iSCREAM, and obtain a ratio of weak keys of 2^{-48} , at the cost of requiring 2^{32} chosen-tweak chosen plaintexts in order to detect whether a key is weak; once a weak key is detected, the full 128-bit key can be recovered in time complexity 2^{48} with no additional data.

Regarding LS-designs, it should be pointed out that while our first attack breaks Robin and iSCREAM in a practical setting, Fantomas and SCREAM appear to be safe. Moreover, in the case of Robin and iSCREAM, a careful tweak of the ciphers should be able to prevent our attacks. Thus, the security of the LS-design framework in general is not called into question. On the contrary, in the case of Zorro, our attack adds to other attacks suggesting that partial nonlinear layers should be approached with caution.

¹ The source code of our tool is available at invariant-space.gforge.inria.fr.

Related Work. Invariant subspace attacks were introduced in [22]. Their application to PRINTCIPHER relies on undesirable properties induced by its 3-bit S-boxes. By contrast, most of our attacks (except the second attack on Robin and iSCREAM) are actually independent of a particular choice of S-boxes.

A thorough analysis of invariant subspaces in PRINTCIPHER was subsequently carried out in [9]. Using a dedicated tool, the authors were able to enumerate all invariant subspaces of PRINTCIPHER, of the type uncovered in the previous article. However their approach is tailored to PRINTCIPHER, and does not extend to other ciphers.

An older line of work has studied “linear factors” of DES [11, 13, 28], which bear some resemblance to invariant subspaces. The existence of a linear factor is an even stronger property than that of an invariant subspace: essentially, it asks that a (linearly defined) portion of the ciphertext only depend on (linearly defined) portions of the plaintext and key. Nonetheless, it is interesting to note that our attacks do uncover a linear factor in Robin and Zorro (the subcipher in Section 3.3), although only in a weak key setting.

Along a similar line, the attack on SAFER in [25] should be mentioned. It exploits the action of the cipher on cosets of a vector space as a whole, rather than isolating a specific trail or characteristic.

Self-similarity properties were used to attack hash functions and block ciphers in [8]. It should be noted that self-similarity is a very wide framework, encompassing attacks ranging from probability one related-key differentials to slide attacks. To the best of our knowledge, the commutation property we consider here is very different from any previous work.

There is no prior cryptanalysis of LS-designs. As for iSCREAM, an issue with the padding in the original CAESAR submission of SCREAM and iSCREAM was pointed out in [29] and subsequently corrected. Our attacks have caused iSCREAM to be temporarily withdrawn from the CAESAR competition for a redesign [17].

By contrast, many attacks have been carried out against Zorro, mostly differential or linear in nature [3, 19, 27, 30, 31]. The best attack in [3] is a differential attack requiring 2^{41} data and time complexity 2^{45} to break the full cipher. Our attack is of a different nature: it holds in the weak key setting (with 2^{96} weak keys out of 2^{128}), requires minimal data and time, and is independent of the number of rounds. Similar to [3], our attack can be readily extended to Zorro-like ciphers, as shown in the ePrint version of this work [23].

Structure of the Paper

In Section 1, we recall the definitions of invariant subspace and present our generic algorithm for detecting such invariant subspaces. In Section 2, we provide a description of LS-designs, including our targets Robin and iSCREAM. In Sections 3, 4 and 5, we develop our attacks against LS-designs, introduce a particular self-similarity property, the resulting invariant subspaces, and finally describe a different invariant subspace attack not underpinned by self-similarity. In Section 6, we apply our self-similarity and invariant subspace attacks to Zorro. Finally, in Section 7, we conclude with a discussion of our results and outline interesting open problems.

1 A Generic Algorithm to Detect Invariant Subspaces

In this section we first recall the invariant subspace attack and later present our algorithmic approach to detect invariant subspaces in a generic manner.

1.1 Invariant Subspace Attacks

Invariant subspace attacks were introduced and applied to PRINTCIPHER in [22]. We briefly recall the basic principle here.

Consider a n -bit block cipher with round function F_K consisting of a key addition and a SP layer $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$. That is, F_K is defined by $F_K(x) = F(x + K)$. Assume the SP-layer F is such there exists a subspace $A \subseteq \mathbb{F}_2^n$ and two constants $u, v \in \mathbb{F}_2^n$ with the property:

$$F(u + A) = v + A$$

Then, given a (round) key $K \in u - v + A$, i.e. $K = K_A + u - v$ with $K_A \in A$, the following holds:

$$F_K(v + A) = F(v + A + u - v) = F(u + A) = v + A$$

i.e. the round function maps the affine subspace $v + A$ onto itself. If all round keys are in $u - v + A$, in particular if identical round keys are used as in LS-designs and Zorro, then this property is iterative over an arbitrary number of rounds.

In the case where an identical key is added in every round (there is no key schedule), a key is said to be weak iff it belongs to $u - v + A$. Whenever a key is weak, plaintexts in $v + A$ are mapped to ciphertexts in $v + A$, breaking plaintext confidentiality. The number of weak keys is the cardinality of A .

In order to detect whether an unknown key is weak, it is enough to encrypt one plaintext in $v + A$, and test whether the resulting ciphertext is in the same space. Indeed, over the set of all keys, false positives will occur with the same frequency as true positives, and can be discarded with a second chosen plaintext.

1.2 A Generic Algorithm

In this section we present a simple and entirely generic probabilistic algorithm able to discover invariant subspaces for a given round function. The algorithm gives instant results for vector subspaces, and is able to discover affine subspaces in time proportional to their density. Despite its simplicity, this algorithm is enough to automatically discover all invariant subspace attacks to be elaborated upon in the following sections.

The algorithm will identify minimal invariant subspaces and thereby identify invariant subspace attacks automatically. However, further analysis usually allows to significantly improve upon the attacks recovered automatically by the algorithm and gain further insights in the structure of the detected weakness. Furthermore, as the expected running time is determined by the density of invariant subspaces, it might well be that not all possible attacks are detected. Thus, for the moment, this generic algorithm cannot be used to fully exclude the existence of invariant subspaces.

Identifying Minimal Subspaces. Assume we are given a permutation $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$. Here F could be a (keyless) round of a block cipher or a cryptographic permutation (like Keccak- f). Our goal is to find affine subspaces $u + A \subset \mathbb{F}_2^n$ such that:

$$F(u + A) = v + A$$

for some $v \in \mathbb{F}_2^n$.

Our algorithm is based on the following trivial observation.

Lemma 1. *Assume $u + A$ is an affine subspace such that $F(u + A)$ is also an affine subspace $v + A$. Then for any subset $X \subseteq A$, the linear span of $(F(u + X) - v) \cup X$ is contained in A .*

The idea is to first guess one possible offset u' of the affine space to be found and use $v' = F(u')$. Next, we guess a one-dimensional subspace of A , denote this by A_0 . The algorithm will succeed if and only if $u' + A_0$ is contained in $u + A$.

1. We compute A_{i+1} from A_i as:

$$A_{i+1} = \text{span}\{(F(u' + A_i) - v') \cup A_i\}$$

2. If the dimension of A_{i+1} equals the dimension of A_i , we found an invariant subspace and exit.
3. If not, we continue with step 1.

Thus, the idea is to start with what we denote *nucleon* of A and map it using F until it stabilizes. In the case that our initial guess was wrong and $u' + A_0$ is not contained in some non-trivial invariant subspace we will end up with the full space after at most n iterations of the above.

Note that it is not necessary to really map the complete spaces A_i using F but a randomly chosen subset of relatively small size is enough for our purpose and significantly speeds up the process.

If the largest invariant subspace of F has dimension d , the algorithm will detect this space (or any invariant subspaces of this space) after an expected number of $2^{2(n-d)}$ guesses for A_0 and u' . Thus, in this basic form, the algorithm becomes quickly impractical. However, in the case of round functions of a cipher (or a cryptographic permutation) that differ by round constants only, its running time can be greatly improved as described next.

Knowing the Nucleon. For block ciphers with identical round keys or cryptographic permutations, we actually have a very good idea about the nucleon we want to be included in the space A , namely the round constants. More precisely, we consider round functions $F_i : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ that differ only by the addition of constants, i.e.

$$F_i(x) = F(x) + c_i$$

for $c_i \in \mathbb{F}_2^n$, where for simplicity we assume $c_0 = 0$. We are looking for affine subspaces $u + A$ that are mapped to $v + A$ by all round functions. In particular

$$F_0(u + A) = F(u + A) = v + A$$

and

$$F_i(u + A) = F(u + A) + c_i = v + A$$

which implies

$$v + A = c_i + v + A$$

and thus $c_i \in A$. Thus, given the situation as above, any subspace that is invariant under all round functions must necessarily contain the linear span of all round constants c_i .

For the algorithm outlined above this has significant consequences. Here, the only thing we have to guess is the offset. Therefore, the expected number of iterations of the algorithm is reduced from $2^{2(n-d)}$ to 2^{n-d} .

Moreover, after running the algorithm for m iterations with randomly chosen guesses for the offset, the probability that an invariant subspace of dimension d is not detected by the approach is given by

$$p_{m,n,d} := (1 - 2^{n-d})^m$$

which can be approximated by

$$\log p_{m,n,d} \approx -m2^{d-n}.$$

The Algorithm.

```

1: procedure CLOSURE(function  $F$ , nucleon  $A$ , offset  $u$ )
2:    $v \leftarrow F(u)$ 
3:   StableCount  $\leftarrow 0$ 
4:   while StableCount  $< N$  do
5:     Pick a random  $x \overset{\$}{\leftarrow} u + \text{span}\{A\}$ 
6:     if  $F(x) - v \in \text{span}\{A\}$  then
7:       StableCount = StableCount + 1
8:     else
9:       Add  $F(x) - v$  to  $A$ 
10:      StableCount  $\leftarrow 0$ 
11:    end if
12:   end while
13:   return  $u + \text{span}\{A\}$ 
14: end procedure
```

For offset u and nucleon A , the above procedure outputs the smallest affine subspace containing $u + \text{span}\{A\}$, that is mapped to a coset of the same space by F (with high probability). The algorithm depends on a global parameter N that controls the risk of error. Namely, when the algorithm exits, elements of $u + \text{span}\{A\}$ are mapped to $v + \text{span}\{A\}$ with probability greater than $1 - 2^{-N}$. This probabilistic result is enough for an invariant subspace attack to go through even for moderate choices of N .

Guessing the Offset. If we are actually looking for stable *vector* spaces rather than affine spaces, as will be the case in the S-box independent setting described in Section 3.2, guessing the offset is not needed: we can choose zero as the offset. Then the algorithm above finds the smallest invariant subspace instantly.

In the general case where we are looking for any (affine) invariant subspace, we need to guess one offset u belonging to the affine space we are searching for. Then we can run the procedure above to find the generated invariant subspace, if it exists (otherwise, the algorithm will simply output the full space). If the space we are looking for has dimension d , guessing such an offset u by brute force will require 2^{n-d} tries on average. Of course we just require *one* invariant subspace; so in general 2^{n-d} can be replaced by the density of vectors belonging to (non-trivial) invariant subspaces.

Each iteration of the algorithm requires Gaussian reduction to determine whether a certain n -bit vector belongs to some subspace, amounting to n^2 operations. Hence the overall running time to find an invariant subspace of dimension d is roughly $n^2 \cdot 2^{n-d}$. Thus if n is large, the above approach will only work if $n - d$ is relatively small, or more generally the density of invariant subspaces is large. The case where n is small is also useful in order to find invariant subspaces through a single S-box: this is how we found spaces in Appendix B (after making the algorithm deterministic and exhaustive, which is affordable for small n).

1.3 Applications

We applied the algorithm to the block ciphers Zorro, Robin, Fantomas, LED and NOEKEON, as well as to the CAESAR candidate iSCREAM. We chose $N = 50$ to be very conservative. We ran the algorithm with approximately 2^{34} iterations for each primitive, stopping earlier in the case where an invariant subspace was detected. The results are summarized in the table below.

Table 1. Experimental Results: Here n is the block size and $d_{0.001}$ is the smallest dimension of an invariant subspace that has a probability to exist upper bounded by 0.001

Primitive	n	Dimension found	$d_{0.001}$	Running Time (h)
LED	64	-	34	24
NOEKEON [12]	128	-	98	40
Fantomas	128	-	98	40
Robin	128	96	-	22
iSCREAM	128	96	-	22
Zorro	128	96	-	<1

For LED, NOEKEON and Fantomas, no invariant subspaces were detected given our limited iterations. In that case, Table 1 indicates the dimension $d_{0.001}$ of the largest invariant subspace that has a probability to exist upper bounded by 0.001. More precisely, if x denotes the codimension of the largest invariant

subspace, each random guess of an offset has probability 2^{-x} of falling into this subspace. After T tries, the probability of not having found the subspace is thus $(1 - 2^{-x})^T \approx e^{-T2^{-x}}$. We want this probability to be 1/1000 within $T = 2^{32}$ tries, which yields $x = 32 - \log(\ln(1000)) \approx 30$, so $d_{0.001} \approx n - 30$. Thus it is unlikely that invariant subspaces of dimension above 98 exist for NOEKEON. However, the existence of smaller subspaces cannot be excluded with high probability by our results.

As we will show below, for Zorro, Robin and the CAESAR candidate iSCREAM the largest invariant subspace has dimension 96 out of 128, i.e. density 2^{-32} . Thus the time complexity is expected to be 2^{32} Gaussian eliminations on 128×128 binary matrices. Our experiments confirm this estimation. Discovering the invariant subspace took 22 hours on a single desktop PC equipped with an Intel Xeon Core i7 with 12 virtual cores used in parallel.

In the case of Zorro, we chose to use a single round as target function, rather than the four rounds separating key addition. It turns out many cosets of the invariant subspace in Appendix A are sent to another coset by a single round (namely, all cosets stemming with offsets where cells 0 and 3 are equal). Our generic approach discovers this fact and the associated subspace instantly, hence the “ < 1 ” time in the previous table.

As mentioned in the introduction, a detailed analysis of the findings of the generic algorithm allows to understand the underlying structure of the invariant subspaces we have found, and improve the attacks. We present those findings in the following sections.

2 Description of LS-Designs, Robin, and iSCREAM

2.1 LS-Designs

LS-designs were introduced by Grosso, Leurent, Standaert and Varici at FSE 2014 [15]. We refer the interested reader to their article for a detailed presentation of LS-designs and their design rationale. For our purpose, a brief technical description suffices.

An LS-design is a block cipher encrypting n -bit plaintext blocks using a n -bit key. The inner state of the cipher, as well as the plaintext, ciphertext, and key, are all represented as an $r \times c$ bit array, with r the number of rows and c the number of columns. A concrete LS-design is parametrized by the following components:

- A choice of r and c . The size of the key and message blocks is $n = r \cdot c$.
- An r -bit S-box s .
- A bijective linear map ℓ on c -bit vectors, called the L-box.
- A number of rounds t .
- A choice of k -bit round constants $C(i)$ for $1 \leq i \leq t$.

In order to encrypt a given n -bit plaintext block, the plaintext is first loaded into the inner state of the cipher, and the master key is added in (all additions

are bitwise XORs). Then a round function is applied successively for rounds 1 to t . At that point the cipherext is equal to the inner state. The round function at round i proceeds as follows:

1. The round constant $C(i)$ is added to the inner state.
2. The S-box s is applied to each column of the state.
3. The L-box ℓ is applied to each row of the state.
4. The n -bit master key K is added to the state.

2.2 Notation

When dealing with LS-designs, we will always use the previous notation; that is:

- r the number of rows of the state.
- c the number of columns.
- n the size of the state; that is, $n = r \cdot c$.
- s the r -bit S-box.
- S the S-box step; that is, the application of s on each column of the state.
- ℓ the $c \times c$ binary matrix representing the linear layer, identified with the corresponding linear map on \mathbb{F}_2^c .
- L the L-box step; that is, application of ℓ on each row of the state.

2.3 Robin

In [15], two concrete LS-designs are proposed, Robin and Fantomas. The idea behind Robin is that both the S-box and L-box are involutive. This allows the same circuitry to be reused when computing these components and their inverse operation, i.e. when encrypting and decrypting. This saves valuable space on embedded devices when both encryption and decryption capabilities are required. The trade-off is that involutive components have more structure, resulting in a slightly higher number of rounds to reach the same security level as an LS-design based on non-involutive components.

Robin strictly fits within the LS-design framework recalled in the previous section. As such it can be fully described by the following parameters:

- The inner state of Robin has 8 rows and 16 columns, resulting in 128-bit blocks and a 128-bit key.
- The 8-bit involutive S-box is given in [15].
- The 16-bit involutive L-box is depicted as a 16×16 binary matrix on Fig. 1.
- The number of rounds is 16.
- At round i (starting from 1), the round constant $C(i)$ is zero outside of the first row, where it is equal to $\ell(i)$, with ℓ the L-box matrix.

2.4 iSCREAM

SCREAM and iSCREAM [18] are two authenticated ciphers closely related to LS-designs. In fact iSCREAM is essentially a tweaked version of Robin, together with a Tweakable Authenticated Encryption (TAE) mode of operation [24].

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
1	■															
2	■															
3																
4																
5	■															
6	■															
7																
8																
9																
10	■															
11	■															
12																
13																
14	■															
15	■															

Fig. 1. Matrix representing the L-box of Robin and iSCREAM. Dark cells stand for 1's and white cells for 0's.

Meanwhile SCREAM is similar to Fantomas, with a different linear layer. The TAE mode of operation requires a tweakable block cipher [24]. Accordingly, the difference between the block cipher underlying iSCREAM and Robin stems from the introduction of a 128-bit tweak T into the (previously non-existent) key schedule.

In the remainder of this article we focus on weaknesses of the block cipher on which iSCREAM is built, independently of the mode of operation. We may abuse notations and write iSCREAM to mean its underlying block cipher.

This block cipher can be described as an LS-design, except for the fact that during the key addition phase, instead of adding in K every round: at odd rounds, $K + T$ is added; while at even rounds, $T \lll_c 1$ is added, where $T \lll_c 1$ denotes a circular shift of the columns of T by one column towards the left. The combination of two rounds is called a step. Beside that, iSCREAM can be described by the following parameters:

- The inner state of iSCREAM has 8 rows and 16 columns, resulting in 128-bit blocks and a 128-bit key.
- The S-box and L-box are those of Robin.
- The number of rounds depends on the required security level. The original article lists six variants. However the primary recommendation for iSCREAM as per CAESAR requirements is 12 steps (24 rounds) [16]. A secondary recommendation claiming related-key security has 14 steps (28 rounds). Since our attacks are essentially independent of the number of rounds, we omit other variants.
- At round i (starting from 1), the round constant $C(i)$ is zero outside of the first row, where it is equal to $27 \cdot i$ modulo 256 (affecting only the first 8 bits of the row).

3 Invariant Permutation Attack

In the next two sections, we analyze the invariant subspace discovered by the generic algorithm on Robin and iSCREAM. This subspace is actually induced by a particular type of self-similarity of independent interest, as is the invariant subspace of Zorro. Using this self-similarity directly results in an even stronger attack, as will be discussed below.

We start by recalling the concept of self-similarity and explaining a link to commutating linear maps. These concepts will afterwards be applied to LS-designs in general and to Robin and iSCREAM in particular (as well as to Zorro in Section 6).

3.1 Self-Similarity Properties and Linear Commutant

In [4] and [8], self-similarity in general is defined as:

Definition 1 (Self-similarity in a block cipher). *For a fixed block cipher E , let $E_K(x)$ denote the ciphertext block resulting from the encryption of plaintext block x under key K . A self-similarity relation is given by invertible and efficiently computable mappings ϕ, ψ, θ such that:*

$$\forall K, x : \quad \theta(E_K(x)) = E_{\psi(K)}(\phi(x))$$

What we are interested in is the case where $M = \phi = \psi = \theta$ is a linear map. This situation will arise if the cipher follows a generalized Even-Mansour structure where key-independent round functions F_i alternate with the addition of a fixed key K (i.e. no key schedule); and M commutes with the round functions F_i . This last condition is very demanding; but this is precisely what happens in both Robin and Zorro, despite their different structure. We expand on why this might be the case in the discussion (Section 7). The following lemma sums up the attack.

Lemma 2. *Consider a block cipher composed of round functions F_i separated by addition of a fixed key K . Suppose there exists a linear map M such that M commutes with the F_i 's. Then:*

$$\forall x : \quad M(E_K(x)) = E_{M(K)}(M(x))$$

In particular, if $K = M(K)$:

$$\forall x : \quad M(E_K(x)) = E_K(M(x))$$

The commutativity of M and the round functions can be interpreted from the invariant subspace perspective. Indeed, if we let $A = \ker(M^i + Id)$ for any i , A is an invariant subspace². Of course self-similarity is a stronger property stemming from a stronger requirement on the cipher.

² It may be that a non-trivial commuting matrix leads only to trivial invariant subspaces, as evidenced by the 2×2 binary matrix with rows [01] and [11]. However if M is involutive, $\ker(M + Id)$ is at least half of the space.

In our applications, M will be involutive, so we focus on the case $i = 1$. In the remainder, whenever two plaintext blocks (or ciphertext blocks, or inner states, or keys) satisfy $x_2 = M(x_1)$, we say that they are *related*. If a plaintext block (or ciphertext block, or inner state, or key) is related to itself, we say that it is *self-related*. A *weak key* is a self-related key. In short, our attack states that weak keys map self-related plaintexts to self-related ciphertexts; while related keys map pairs of related plaintexts to pairs of related ciphertexts.

3.2 S-box-Independent Setting

We now focus on the case where the cipher is a substitution-permutation network (SPN), whose round function F_i consists of an S-box layer with identical S-boxes, a linear map L , addition of a round constant $C(i)$, and addition of a fixed key K . From the invariant subspace (resp. self-similarity) perspective, we are interested in subspaces (resp. linear maps) that traverse (resp. commute with) each of these components.

It is quite apparent that the main roadblock is the non-linear S-box layer. However even in a generic setting where we do not take into account a particular choice of S-box, any permutation of the S-box inputs will commute with the S-box layer (due to S-boxes being identical). Thus we restrict our attention to permutations of S-box inputs rather than general linear maps.

In terms of invariant subspaces, this corresponds to subspaces containing those vectors whose coordinates belonging to the same cycle in the permutation are equal; that is, subspaces that only require S-box inputs to be equal to some other input, or independent. We call such spaces *equality spaces*. Note that these are vector subspaces and no longer affine subspaces. Our strongest attacks actually occur in this setting.

As for constant and key addition, asking that their addition commutes with M amounts to asking that they belong to $\ker(M + Id)$. Now it remains to find permutations that commute with the linear layer. An efficient algorithm to do so is provided in the ePrint version of this work [23]. The invariant subspace variant seems more difficult, as we do not know an algorithm able to efficiently enumerate equality spaces that traverse a linear map.

3.3 Key Recovery

The self-similarity attack above breaks plaintext confidentiality. In addition, if the commuting permutation P is involutive (as will be the case in our applications), efficient key recovery may be possible. In short, the part of the key corresponding to fixed points of the permutation can be guessed independently of the rest.

Intuitively, this is because if two self-related inner states differ only outside the fixed points of the permutation P , this difference will never be propagated to the fixed points of P . This is clear for the S-box layer (because the permutation operates on entire S-box inputs), but also holds for the linear layer. A general

statement and proof are provided in the ePrint version of this work [23]. In fact the proof also encompasses the case where the S-box layer is partial.

What we show is that the cipher contains an embedded subcipher operating on the fixed points F of the permutation: we can project self-related plaintexts and ciphertexts on F and obtain a well-defined map. Note that this embedded subcipher may lend itself to further attacks; this is a direction we have not investigated, as we believe ciphers are sufficiently broken at that point.

3.4 Invariant Permutation Attack on LS-Designs

Notation. In the S-box-independent setting of Section 3.2, for an LS-design, a permutation of S-box inputs is simply a permutation of the columns of the state. Let us write P for such a permutation. We always denote by the lowercase p its effect on a single row. Thus, P is the application of p on each row of the state. We identify p with the corresponding $c \times c$ permutation matrix. We adopt notations from Section 2.2.

The particular structure of LS-designs means that P commutes with L iff p commutes with ℓ . This is still a strong requirement, but we expect the L-box of an LS-design to have some structure in order to provide a good branch number, especially if it is involutive. In the case of Robin for instance, the linear layer is built from a Reed-Muller code and provides plenty of structure. Applied to LS-designs, Lemma 2 becomes:

Lemma 3. *For an LS-design, assume there exists a permutation P with the following properties:*

- P commutes with L .
- $P(C(i)) = C(i)$ for all round indices i .

Then for any plaintext message m :

$$\text{Enc}_{P(K)}(P(m)) = P(\text{Enc}_K(m))$$

In particular, if $K = P(K)$:

$$\text{Enc}_K(P(m)) = P(\text{Enc}_K(m))$$

Note that the identity permutation trivially satisfies the above requirements. Hereafter we always assume P is non-trivial. If $\text{ncycles}(p)$ is the number of cycles of p , weak keys form a proportion $2^{-r \cdot (c - \text{ncycles}(p))}$ of all keys (namely, those keys whose columns are equal on each cycle of p).

Key Recovery. The previous attack breaks plaintext confidentiality. In addition, when P is involutive, efficient key recovery is possible, as announced in Section 3.3. A general statement and proof are provided in the ePrint version of this work [23].

It may still be worthwhile to provide a simpler statement dedicated to LS-designs. This is what we propose below.

Lemma 4. Consider an LS-design, and assume there exists a permutation P with the same requirements as in Lemma 3. Also assume that P is an involution. Consider a weak key $K = P(K)$. Denote by F the set of fixed points of P .

Take any self-related plaintext $m = P(m)$. Then the value of the ciphertext $\text{Enc}_K(m)$ on the columns in F only depends on the value of m and K on the same columns.

Proof. Since P is an involution, all of its cycles have length 1 or 2. Hence we can partition the columns of the state into three subsets F , A , B , such that P is the identity on F , and maps A and B into each other. Take any self-related message m that is zero on F . Then the linear layer maps m to a self-related state $L(m)$ that is also zero on F . To see this, write $m = m_A + m_B$, where m_A is equal to m on A , and zero elsewhere, and likewise m_B is equal to m on B and zero elsewhere. Then $P(m_A) = m_B$, hence $P(L(m_A)) = L(m_B)$ by commutativity of P and L . Since P is the identity on F , this implies that $L(m_A) + L(m_B)$ is zero on F , so $L(m)$ is zero on F .

Thus, if $m = P(m)$ is zero on F , so is $L(m)$. By linearity, this implies that if m_1 and m_2 are self-related and equal on F , then so are $L(m_1)$ and $L(m_2)$. Thus, the property that two self-related states are equal on F goes through the linear layer. This property automatically goes through the S-box layer since it is column-wise. Since the same key and round constants are added to both sides, they have no impact. Hence this property goes through the whole cipher. \square

As a direct consequence, the value of the key on the columns corresponding to fixed points of P can be guessed independently of the rest of the key by using any self-related plaintext. In addition, the embedded subcipher is a smaller LS-design, and may lend itself to further attacks. As a side note, both this lemma and the previous one also show that the cipher is malleable in a strong sense.

Permutation Characteristic. Instead of considering only permutations P commuting with L , we can naturally look for pairs of permutations (P, Q) such that $L \cdot P = Q \cdot L$. We denote this by $P \rightarrow Q$, representing the fact that if two inner states are related by P before the linear layer, then after the linear layer they are related by Q .

From there we can hope to build a form of characteristic $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow \dots$. The commutative case in the previous section corresponds to $P \rightarrow P$. Note that the set of permutations P such that $Q = L \cdot P \cdot L^{-1}$ is a permutation forms a group. Also note that if L is involutive, $P \rightarrow Q$ is equivalent to $Q \rightarrow P$: indeed $L \cdot P = Q \cdot L$ implies $P^{-1} \cdot L = L \cdot Q^{-1}$, implies $L \cdot Q = P \cdot L$: hence any transition $P \rightarrow Q$ yields an iterative characteristic of length at most 2.

A particularly interesting case occurs whenever $P \rightarrow P^\alpha$ for some $\alpha \neq 0$. Indeed, in that case we automatically have a cyclic characteristic $P \rightarrow P^\alpha \rightarrow P^{\alpha^2} \rightarrow \dots \rightarrow P^{\alpha^i} = P$. Moreover the attack from Lemma 3 goes through with exactly the same requirements on the key and round constants (namely they are self-related by P).

Application to Robin. Applying our attack to Robin amounts to finding a permutation p commuting with the matrix ℓ in Fig. 1, such that P leaves all round constants $C(i)$ invariant. More generally, as pointed out just above, we can actually look at transitions $P \rightarrow Q$, i.e. permutations p, q such that $\ell \cdot p = q \cdot \ell$. It turns out there are 720 such transitions, and all of them are of the form $P \rightarrow P^{-1}$. Moreover 76 of these permutations are involutive, and hence commute with L .

Recall that the round constants of Robin are defined as $C(i) = \ell(i)$ on the first row, and zero on the others, for $1 \leq i \leq t$. Hence we want $p(\ell(i)) = \ell(i)$, which amounts to $p(i) = i$ by commutativity. Since i ranges from 1 to 16, what we are looking for is simply permutations leaving the first 5 columns fixed. It turns out there exists exactly one such permutation, namely the involutive permutation P_0 switching columns 8, 9, 10, 11 respectively with columns 12, 13, 14, 15. Looking at Fig. 1, one can indeed see that permuting the rows and columns of the matrix of ℓ by p_0 leaves the matrix invariant, which is the same as saying p_0 commutes with ℓ .

With P_0 , weak keys are simply keys whose last four columns are equal to the previous four. In particular the proportion of weak keys is 2^{-32} . Furthermore P_0 leaves the first 8 columns fixed, so Lemma 4 shows that for self-related plaintexts, the first 8 columns of ciphertexts only depend on the first 8 columns of plaintext and key. This makes it possible to guess the value of the master key on the first 8 columns independently of the rest of the key. This means 64 bits of the key can be guessed separately; then the remaining 64 bits are symmetric through P_0 , so only 32 bits remain to be guessed. Thus the full key can be recovered in time complexity 2^{64} by encrypting any self-related message. This may yield a few solutions, which can be checked against any other plaintext/ciphertext pair.

Table 2. Permutations p_0 , p_1 and p_2 . Fixed points are omitted.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p_0									12	13	14	15	8	9	10	11
p_1							8	9	10	15	4	5	6			7
p_2							12	13	14	11			7	4	5	6

Beside P_0 , two other permutations P_1 and P_2 commuting with L leave the round constants invariant up to the very last round (cf. Table 2). This means related plaintexts are mapped to related inner states after 15 encryption rounds; followed by the final constant addition, S-box layer, L-box layer, and key addition. The final linear layer can be reversed, and the resulting states will agree on pairs of columns transposed by P on which $C(16)$ is equal. In both cases, there is one such pair, so self-related keys with respect to P_1 and P_2 can still be detected easily by encrypting a few self-related plaintexts, reversing the last linear layer, and checking that these two columns agree.

Permutations P_1 and P_2 both leave 8 columns fixed and hence yield an attack with essentially the same properties as P_0 . Actually some key bits can be recovered faster than with P_0 thanks to the one-round differential at the end, but this

involves the symmetric part of the key (that is, outside the fixed points of the permutation) and thus the overall key recovery time is still 2^{64} .

Application to iSCREAM. Recall that iSCREAM and Robin share the same linear layer. Round constants only affect the first eight columns of the state, and so we are looking for permutations commuting with L and leaving the first eight columns unchanged. As a matter of fact, there exists exactly one such permutation, namely the same permutation P_0 as above, which switches the last four columns of the state with the previous four.

Another difference between Robin and iSCREAM is the number of rounds, but that is actually irrelevant for our attack. The last difference is the presence of a tweak in the key schedule. Recall that at odd rounds, $T + K$ is added, while at even rounds, $T \lll_c 1$ is added, where T is a 128-bit tweak. In a chosen-tweak scenario, we can simply set T to zero, or any other value such that T and $T \lll_c 1$ are invariant by P . Then the attack against Robin from the previous section applies to iSCREAM essentially unchanged, with the same consequences.

A small variant of our attack is also possible when using P_0 as the commuting permutation. What we truly want is that $K + T$ and $T \lll_c 1$ should be self-related. This amounts to asking that columns 8, 9, 10, 11 should be equal to columns 12, 13, 14, 15. Since $T \lll_c 1$ is a column-wise shift of T by one column towards the left, this means that columns 9, 10, 11, 12 of T should be equal to columns 13, 14, 15, 0. Note that there is no condition on column 8 of T . As a consequence, for $K + T$ to be self-related for some choice of T , it is enough to ask that columns 9, 10, 11 of K should be equal to columns 13, 14, 15. Indeed in that case, we can fix T to be all-zero, except for column 8 which can take any value: exactly one such choice of T will satisfy that $K + T$ is self-related. Thus we obtain a larger set of weak keys (with ratio 2^{-24}), at the cost of requiring 2^8 chosen-tweak messages in order to detect whether a fixed unknown key is weak.

In addition, some variants of iSCREAM claim related-key security. If two keys are related by P_0 , then our attack applies immediately without any weak key requirement, following the first consequence of Lemma 3. That is, related plaintexts are mapped by the related keys to related ciphertexts. Thus it is easy to check whether a pair of keys is related, and the cipher is broken in a strong sense.

Generalizations of the Permutation Attack. There appears to be a few simple ways in which our attack could be generalized. We discuss them briefly here.

We could consider a probabilistic version of the attack. Instead of requiring $L \cdot P = P \cdot L$, we could consider P 's such that the kernel of $L \cdot P - P \cdot L$ is almost the full space. In the case of Robin or iSCREAM, this would incur a cost at least 2^{-8} per round.

Another natural extension is to consider cases where all round constants are P -invariant except for the last few rounds (or first few rounds). Then our attack goes through most of the encryption process, and eventually yields a differential

attack on the remaining rounds. When encrypting self-related plaintexts, this differential attack turns into an inner differential.

4 Invariant Equality Space Attack

In this section we study invariant subspaces for LS-designs following the S-box-independent setting of Section 3.2. We begin by defining equality spaces, and then present our results on Robin and iSCREAM. On the way, we will recover the invariant subspace detected by our generic algorithm (Section 1.3), and link it to the commuting permutations from the previous section.

4.1 Equality Spaces

As always, we use notations from Section 2.2. We always view n -bit vectors as an $r \times c$ matrix. In Section 3.2, we defined equality spaces in general terms for an SPN; we now provide a more specific definition suited to LS-designs.

Definition 2. A subspace E of $\{0, 1\}^c$ is an equality space iff there exists a partition of $\{0, \dots, c - 1\}$ such that E is the set of vectors whose values on coordinates belonging to the same class in the partition are equal.

The dimension of E is the number of classes of the partition. By E^r we denote the set of n -bit states whose columns belonging to the same class in the partition underlying E are equal. Equivalently, this means that every row of the state belongs to E , hence the notation E^r . By extension we also call E^r an equality space. The point of this definition is that equality spaces are preserved by the S-box layer. The question is to determine which equality spaces are also preserved by the linear layer. That is, we are looking for equality spaces $E \subset \{0, 1\}^c$ such that $\ell(E) = E$.

As pointed out in Section 3.1, when a permutation P commutes with L , the equality space defined by the cycles of P is preserved by the linear layer. The idea is that equality spaces preserved by the linear layer do not necessarily stem from a commuting permutation. Conversely, commuting permutations are an interesting special case, since they lead to a stronger property: indeed, when considering equality spaces rather than permutations, we are looking at a property of a single state, and there is no equivalent to the property that distinct related plaintexts are mapped to related ciphertexts; there is also no equivalent to Lemma 4. Meanwhile, Lemma 3 becomes:

Lemma 5. For an LS-design, assume there exists an equality space E such that:

- $\ell(E) = E$.
- $C(i) \in E^r$ for all round indexes i .

Then for any key K and plaintext message m :

$$\text{If } K \in E^r \text{ and } m \in E^r \text{ then } \text{Enc}_K(m) \in E^r$$

The lemma trivially holds if E is the full space $\{0, 1\}^c$; hereafter we assume this is not the case. Then we have an attack in the weak key setting, where weak keys are keys in E^r . Hence the proportion of weak keys is $2^{-r \cdot (c - \dim(E))}$.

4.2 Variants of the Attack

Essentially the same extensions as in Section 3.4 apply to equality spaces.

Characteristics: if the image $F = L(E)$ of an equality space E is also an equality space, we write $E \rightarrow F$. As with permutations, we can aim to build a characteristic $E_0 \rightarrow E_1 \rightarrow \dots$ over several rounds. Note that the set of equality spaces is closed under intersection, and as a direct consequence, the set of equality spaces E such that $L(E)$ is an equality space is also closed under intersection. If L is involutive, $E \rightarrow F$ is equivalent to $F \rightarrow E$, so characteristics are automatically cyclic.

Probabilistic attack: Instead of asking $F = L(E)$, we can require the dimension of the quotient space $F/L(E)$ to be small.

Differential ending: If all round constants are in the required equality spaces except for the last few (or first few) rounds, it may be possible to cover the remaining rounds with an inner differential characteristic. Indeed in the case $E \rightarrow E$, the equality space attack may be seen as an all-zero inner differential attack.

Differential attack: the entire attack itself may be transposed into the differential world, at the expense of becoming probabilistic. Consider a state difference living in E^r with $L(E) = E$. Then at each round, require that the S-box layer preserves this equality; that is, the output of some S-boxes which receive equal input, should remain equal. Note that if E stems from an involutive permutation commuting with L , the columns corresponding to fixed points of the permutation can be set to a zero difference: this will be preserved by the linear layer (cf. the proof of Lemma 4). This attack avoids key and round constant requirements, at the cost of much lower probability, and hence high data requirements. In practice this would lead to a weaker attack against Robin than truncated differential product trails in the original article [15], because the branch number is 8 and the non-fixed points of P_0 involve 8 S-boxes.

4.3 Application to Robin and iSCREAM

Since Robin and iSCREAM share the same linear layer L , we consider them together. We enumerated all equality spaces E such that $L(E)$ is an equality space (there are around 2^{33} partitions of 16 elements, so this is feasible), and analyzed the results.

Our first observation is that there are many more *well-behaved* equality spaces E (in the sense that $L(E)$ is also an equality space), than *well-behaved* permutations P (in the sense that $Q = L \cdot P \cdot L^{-1}$ is also a permutation). Namely, there are 720 well-behaved permutations for L , while there are 30162 well-behaved spaces of dimension 8 or more. Even if we remove from this list spaces that are an intersection of larger well-behaved spaces (and thus could have a chance of indirectly resulting from well-behaved permutations), 7746 well-behaved spaces remain.

Recall that L is involutive, so any transition $E \rightarrow F$ (i.e. $L(E) = F$ with E and F two equality spaces) yields a cyclic characteristic $E \rightarrow F \rightarrow E$. Hence all

well-behaved spaces belong to cycles of length 1 or 2. The aforementioned 7746 intersection-reduced well-behaved spaces of dimension at least 8 form 2506 cycles of length 1 (that is, $E \rightarrow E$) and 2620 cycles of length 2 (that is $E \rightarrow F \rightarrow E$). Thus equality spaces offer considerably more potential attacks, depending on round constants.

However, all equality spaces compatible with actual round constants for Robin minus the last round, and hence directly usable in an attack, stem from commuting permutations. There exist four such spaces: three of them correspond to permutations P_0 , P_1 and P_2 from Table 2, and the last one is a space of dimension 8 resulting from the composition of any two of the previous permutations (any combination yields the same permutation or its inverse). As for iSCREAM, the only well-behaved space compatible with round constants is the one resulting from P_0 . Thus, our previous attack is not improved. Moreover, the largest well-behaved spaces have dimension 12 and all stem from involutive permutations (there are 15 of them). The largest well-behaved equality spaces not stemming from a well-behaved permutation have dimension 10. This may be interpreted to mean that the strongest phenomenon is due to commuting permutations.

Thus for both Robin and iSCREAM, the equality space induced by P_0 is the only equality space that goes through the whole cipher, including the last round. This space has dimension 96 over \mathbb{F}_2 , and it is the invariant subspace automatically discovered by the generic algorithm from Section 1.

4.4 A note on Fantomas and SCREAM

The matrix L of Fantomas is a permutation of the lines and columns of the matrix of Robin. As a consequence, they have the same number of well-behaved permutations and spaces. However we found no cycle among well-behaved spaces of Fantomas of dimension 6 or more (lower dimensions would yield very weak attacks); and no characteristic of length more than 2. Hence Fantomas seems safe from this attack.

The same is true for SCREAM. However, it is worth noting that there exists no well-behaved permutation for the matrix of SCREAM, while we found 5404 well-behaved spaces of dimension 8 or more.

5 A Second Invariant Subspace Attack on LS-Designs

In this section we present a different invariant subspace attack on LS-designs, which may be regarded as a form of dual of the previous attack. This attack does not stem from an underlying permutation; nor does it have an equivalent for Zorro. Thus, this section is specific to LS-designs, and takes advantage of their particular structure: namely, the fact that LS-designs not only rely on a layer of identical S-boxes, but also on a layer of identical L-boxes.

Now that we have understood the invariant subspace discovered by our generic algorithm as being an equality space, i.e. a space that is automatically preserved by the S-box layer, it is natural to ask if something similar can be

done with the L-box layer. That is, we are now going to look for a property that is automatically preserved by the L-box layer.

This gives us more freedom, since we can leverage linearity. Essentially, if all columns of the state live in the same linear subspace, this will remain true after the linear layer (in the ePrint version of this work [23], we prove that this is in fact the most general property generically preserved by the linear layer); whereas in the previous case, we were limited to equality spaces. Beside this difference, the attack is essentially a dual version of the previous one, reversing the roles of the L-box and S-box layers.

5.1 Description of the Attack

In the previous attack, we searched for equality spaces $E \subset \{0, 1\}^c$ on the rows of the state such that $\ell(E) = E$. Instead, we are now interested in general linear subspaces $A \subset \{0, 1\}^r$ on the columns of the state such that $s(A) = A$. Once again, if A is a linear space on the columns (or one of its cosets), we denote by A^c the set of states whose columns all belong to A .

The core of the attack is the following: assume $s(A) = A$ for some linear space A . If the inner state lies in A^c , this will remain true after the S-box layer. Moreover, this property is automatically preserved by the linear layer. Indeed, the linear layer of an LS-design is not truly “line-wise”: precisely because the same linear map is applied to each row, the linear layer may be seen as directly adding together column vectors. From this point of view, it becomes clear that if all columns lie in the same linear space A , this remains true after the linear layer.

Thus we are still within the invariant subspace framework, and follow the corresponding strategy: we choose A such that all round constants belong to A^c , and we consider a weak key scenario by requiring that the key also lie in A^c . If these requirements are fulfilled, plaintexts in A^c are mapped to ciphertexts in A^c .

More generally, we can consider cosets of linear spaces (i.e. affine spaces) rather than just linear spaces: indeed, as long as each coordinate at the output of ℓ is the sum of an odd number of coordinates at the input, the linear layer still preserves the property that all columns belong to a fixed coset. The following lemma sums up the attack.

Lemma 6. *Let u, v, w be r -bit vectors, and A be a linear subspace of r -bit vectors. Assume the following conditions hold:*

- The S-box s maps all vectors in $u + A$ to vectors in $v + A$.
- Either $v = 0$ or all rows of the matrix of ℓ have an odd number of 1's.
- The columns of all round constants are in $w + A$.
- The columns of the key are in $(u + v + w) + A$.

Then any plaintext in $(u + w) + A$ is encrypted into a ciphertext in $(u + w) + A$ (and conversely).

Weak keys are keys in $(u + v + w) + E$. This means a proportion $2^{-c \cdot (r - \dim(A))}$ of keys is weak.

5.2 Application to Robin and iSCREAM

In the case of Robin, the second condition in Lemma 6 is automatically true. In order to satisfy the third condition (round constants), since round constants only affect the first row of the state, we require that the r -bit vector denoted by 1, with 1 on the first row and 0 elsewhere, belongs to E . To instantiate the attack, it remains to look for affine spaces whose direction contains the vector 1, that are mapped by the S-box to affine spaces with the same direction.

It turns out the largest such spaces have dimension 3, and are mapped into themselves. We list all six choices in Table 3. Since these spaces have dimension 3, and the state has 8 rows and 16 columns, a proportion $2^{-16 \cdot 5} = 2^{-80}$ of keys are weak. This means our attack is considerably weaker than the first one against Robin. By comparison, a generic multi-target time-memory trade-off with 2^{48} memory would lead to key recovery for the same proportion of keys. Of course our attack requires no memory or table lookup.

Table 3. Six affine spaces of dimension 3 invariant through s

Values in A									Dir(A)		
00	01	26	27	84	85	a2	a3	01	26	84	
18	19	7c	7d	9e	9f	fa	fb	01	64	86	
28	29	32	33	8a	8b	90	91	01	1a	a2	
3c	3d	5e	5f	b2	b3	d0	d1	01	62	8e	
44	45	66	67	c8	c9	ea	eb	01	22	8c	
4e	4f	54	55	6c	6d	76	77	01	1a	22	

We now turn to iSCREAM. Recall that its S-box is the same as that of Robin, and round constants still only affect the first row of the state. We want both $K + T$ and T to live in the same coset, so we require T to lie in $(u + v + w + A)^c$, and K to lie in A^c . In our actual attack we have $u = v$ and $w = 0$ so in the end, we can set the tweak to zero (or any value in A^c), and the attack goes through with the same parameters as before.

5.3 Taking Advantage of the iSCREAM Tweak Schedule

In the case of iSCREAM, it is possible to leverage the tweak schedule to create a trade-off between the ratio of weak keys and the number of chosen-tweak messages required to detect a weak key. To simplify notations, we explain this technique using vector spaces; it extends to their cosets in a straightforward manner. Assume we have two vector spaces A and B with $S(A) = B$. As before, we assume $1 \in A$ and $1 \in B$ so that round constants belong to A^c and B^c . Since S is involutive, we have $S(B) = A$, so $A \rightarrow B \rightarrow A$ is a characteristic for the the S-box.

In order for this characteristic to traverse encryption, we need $K + T \in A^c$, and $T \lll_c 1 \in B^c$, which is equivalent to $T \in B^c$. For this it is enough to ask $K \in A^c + B^c = (A + B)^c$. Indeed in that case, write $K = K_A + K_B$ with

$K_A \in A^c$ and $K_B \in B^c$. Then for $T = K_B$, we have $K + T \in A^c$ and $T \in B^c$, which is precisely what we want. Of course the key is unknown to the attacker, so she cannot compute T in this way. Instead, she can try every value in the supplementary space of A^c in $(A+B)^c$ (which is smaller than B^c , if only because $1 \in A \cap B$). For exactly one such value of the tweak, every plaintext in A^c will be encrypted to a ciphertext in B^c .

Now the question is to find two spaces A and B as above. Actually we look for cosets of linear spaces with the same properties, since the linear layer of iSCREAM also preserves these cosets. In summary, we look for affine spaces $u + A \neq v + B$ such that $S(u + A) = v + B$, and 1 belongs to $A \cap B$.

It turns out the largest such spaces have dimension 3. There are 11 such spaces (counting only 1 for $u + A \rightarrow v + B$ and $v + B \rightarrow u + A$), listed in Appendix B. Furthermore, 8 of these spaces satisfy $\dim(A + B) = 5$, which is the maximal possible value since 1 belongs to $A \cap B$. Thus $K \in (A + B)^c$ yields a ratio of weak keys of $2^{-c \cdot (r - \dim(A + B))} = 2^{-48}$.

In order to detect whether a key is weak, one needs to encrypt a message for each tweak in the supplementary of A^c in $(A + B)^c$, which is of dimension $2 \cdot c$, hence 2^{32} chosen-tweak messages are required (for a random key and a given choice of the tweak, a false positive has probability only 2^{-80} , and can be discarded by one additional chosen-tweak message). Finally, once a weak key is detected in this way, we know $K + T \in A^c$ for one specific T , hence $K = T + A^c$, so only $2^{c \cdot \dim(A)} = 2^{48}$ possibilities remain for the value of the key.

5.4 Variants of the Attack

It seems natural to consider a probabilistic version of the attack, where instead of requiring that every vector in $u + A$ be mapped by the S-box to a vector in $v + A$, we only require *most* of them to comply. If only x elements in $u + A$ are not mapped to $v + A$, the probability to pass an S-box is $1 - x/2^r$. The cost for each round is then $(1 - x/2^r)^c$. In the case of Robin, there is no A of dimension 4 with $x < 3$, so there does not appear to be an obvious interesting probabilistic version of the attack.

6 Commuting Permutation and Invariant Subspace for Zorro

6.1 Description of Zorro

The block cipher Zorro was introduced at CHES 2013 [14]. Like LS-designs, the design goal is to offer a cipher that can efficiently be made resistant to side-channel attacks through masking [26]. This is achieved by two main techniques: first, a carefully constructed 8-bit S-box; and second, an AES-like structure where S-boxes are only applied on the first row of the state.

The 128-bit state is represented as a 4×4 array of 8-bit cells. The round function applies the following transformations:

- **SubBytes**: A fixed 8-bit S-box is applied to the first row of the state.
- **AddConstant**: At round i , the constants i , i , i and $i \ll 3$ are added to the four cells of the first row (from left to right).
- **ShiftRow**: This step is identical to AES. Row i , counting from zero, is shifted by i cells to the left.
- **MixColumns**: This step is again identical to AES. A fixed 4×4 circulant matrix on \mathbb{F}_{2^8} is applied to each column of the state. The matrix is the same as that of AES.

Four consecutive rounds are called a step. After each step, the 128-bit master key is simply added to the inner state: there is no key schedule. Encryption consists in key addition, followed by 6 steps (24 rounds), each followed by key addition.

6.2 Self-Similarity and Invariant Subspace

We are interested in an S-box-independent commuting linear map, as in Section 3.1. To simplify, we focus on a single round: commuting with every round is a sufficient condition to commute with every step. Thus we are looking for a linear map M acting as a permutation on the S-boxes, and commuting with the linear layer.

Since there are only four S-boxes, there are only 24 choices for the permutation. In fact, because the constant added to the fourth S-box is different from the others, we impose that this S-box should remain fixed by the permutation, leaving only 6 possibilities. In this way, our linear map will automatically commute with both the S-box and constant addition layers.

For each of the 6 permutation choices on the first 3 S-boxes, the set of linear maps behaving as this particular permutation on the first 4 cells, and independently on the other cells, is itself a vector space. Furthermore the commutant of the linear layer is naturally a vector space. Thus, it suffices to intersect these two spaces to find a solution, if it exists.

It turns out there exists exactly one solution, for the permutation swapping the first and third S-boxes, and leaving the other two fixed. This solution is given in Appendix A, together with the resulting invariant subspace. This subspace has dimension 12 over \mathbb{F}_{2^8} , that is, 96 over \mathbb{F}_2 . Hence the proportion of weak keys is 2^{-32} .

In the ePrint version of this work [23], we show how to enumerate all invariant subspaces for Zorro, and deduce that the previous space is in fact the only invariant subspace (in the S-box-independent setting). The strategy used to enumerate spaces extends naturally to any SPN with a partial S-box layer of only a few S-boxes per round.

6.3 Key Recovery

The key recovery strategy from Section 3.4 extends to partial S-box layers such as Zorro. In brief, if an involutive linear map commutes with the components of an SPN, and acts as a permutation on the S-box inputs, part of the key may

be recovered independently of the rest. When the S-box layer is full, i.e. the commuting map is simply a permutation, this part of the key corresponds to the fixed points of the permutation. When the S-box layer is partial, and hence the commuting map M is not fully a permutation, the role of the non-fixed points is essentially played by $I = \text{Im}(M + Id)$. A formal statement and proof are provided in the ePrint version of this work [23].

The consequence for Zorro is that once a key is recognized as weak, 64 bits of the key can be guessed independently of the rest using one chosen plaintext (any self-related plaintext). Indeed, the part of the key in I only influences the part of the ciphertext in I . After these 64 bits have been recovered by brute force, only 32 bits remain to be guessed, due to the key being weak. Thus key recovery requires only one chosen plaintext and a time complexity of 2^{64} offline encryptions.

7 Conclusion

In this article, we present a unified cryptanalysis of several ciphers based on invariant properties traversing the cipher under certain conditions, while providing generic tools for this type of attack. Our attacks are able to break lightweight ciphers Robin, iSCREAM and Zorro in a practical setting.

Our attacks from sections 4 and 5 are quite similar in principle. The state of an LS-design is a rectangular array. A fixed line-wise operation is performed in each direction. Each attack looks for properties of the inner state that would be *structurally* preserved in one direction (in the sense that this does not depend on the specificities of the S-box or linear layer), that would happen to also be preserved in the other (this time due to the particular choices of S or L).

In the case where the generic direction is linear, any linear space is preserved, and under some conditions any coset; if it is nonlinear, only equality spaces are preserved. In the ePrint version of this work[23], we prove that these are in fact the most general properties structurally preserved in each direction, so our attacks fully realize the program outlined in the previous paragraph. It remains an open question whether a similar attack could in some way combine information from both directions; that is, neither direction would preserve the invariant property in a fully generic way.

Concerning our first attack on LS-designs from sections 3 and 4 (encompassing both invariant permutations and invariant equality spaces), the structure of the linear map is a key component. It seems unlikely that the attack could succeed in cases where the linear layer is not involutive. Indeed, as shown by the matrices of SCREAM and Fantomas, even in the presence of a large number of well-behaved equality spaces, it appears that iterative characteristics do not occur by accident. By contrast, if the linear layer is involutive, any well-behaved equality space (or permutation) yields a cyclic characteristic of length at most 2;

and indeed, in the case of Robin and iSCREAM, thousands of iterative characteristics exist. Of course, the matrix of Robin and iSCREAM has much more structure than a generic involutive matrix.

It is quite striking that exactly the same attacks exist on Zorro, despite its quite different structure (byte-oriented vs. bit-oriented, partial S-box layer vs. full, AES-like vs. somewhat SERPENT-like). It is worth noting however that both ciphers attempt precisely the same goal, namely to offer efficient masked implementations. As a result both reduce non-linear operations to a minimum per round, while giving more weight to the linear layer; LS-designs achieve this by parallelizing the S-box through bit slicing; Zorro by resorting to a partial S-box layer. In both cases the contribution of the non-linear layer is very structured with respect to the linear layer; this, together with the minimal key schedule and simple round constants leads to our attacks.

We note that all our attacks can be prevented by a careful choice of round constants. One needs only ensure that no weaker (such as probabilistic or differential) version of the attack is left behind. This is particularly true when claiming related-key security (as in iSCREAM), since in this setting our attacks do not require weak keys, and hence weaker probabilistic versions are quite relevant.

Going back to the generic algorithm used to find the attacks, an interesting open problem is to specialize it to SPN structures, hoping to achieve better time complexity. In particular, it may be worthwhile to find an algorithm that is able to enumerate all invariant subspaces through a layer of n S-boxes, given n and the S-box. With improvements in time complexity, it may become possible to entirely disprove the existence of invariant subspaces for some SPNs.

Finally, we hope our analysis contributes some insight for the design of future ciphers with minimal key schedules and the choice of round constants in cryptographic permutations.

Acknowledgments. The authors would like to thank Henri Gilbert for many fruitful discussions related to the attacks presented in this article.

A Commuting Linear Map and Invariant Subspace for Zorro

The commuting linear map M is represented as a 16×16 matrix over \mathbb{F}_{2^8} , using the AES representation of \mathbb{F}_{2^8} as $\mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$.

$$\left[\begin{array}{cccccccccccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 34 & 101 & 35 & 101 & 50 & 249 & 50 & 249 & 249 & 116 & 249 & 116 & 116 \\ 0 & 0 & 0 & 0 & 101 & 35 & 101 & 34 & 249 & 50 & 249 & 50 & 116 & 249 & 116 & 249 \\ 0 & 0 & 0 & 0 & 35 & 101 & 34 & 101 & 50 & 249 & 50 & 249 & 249 & 116 & 249 & 116 & 116 \\ 0 & 0 & 0 & 0 & 101 & 34 & 101 & 35 & 249 & 50 & 249 & 50 & 116 & 249 & 116 & 249 \\ 0 & 0 & 0 & 0 & 17 & 86 & 17 & 86 & 1 & 0 & 0 & 0 & 0 & 249 & 50 & 249 & 50 \\ 0 & 0 & 0 & 0 & 86 & 17 & 86 & 17 & 0 & 0 & 0 & 0 & 1 & 50 & 249 & 50 & 249 \\ 0 & 0 & 0 & 0 & 17 & 86 & 17 & 86 & 0 & 0 & 1 & 0 & 0 & 249 & 50 & 249 & 50 \\ 0 & 0 & 0 & 0 & 86 & 17 & 86 & 17 & 0 & 1 & 0 & 0 & 0 & 50 & 249 & 50 & 249 \\ 0 & 0 & 0 & 0 & 51 & 190 & 51 & 190 & 86 & 17 & 86 & 17 & 35 & 101 & 34 & 101 & 101 \\ 0 & 0 & 0 & 0 & 190 & 51 & 190 & 51 & 17 & 86 & 17 & 86 & 101 & 34 & 101 & 35 & 35 \\ 0 & 0 & 0 & 0 & 51 & 190 & 51 & 190 & 86 & 17 & 86 & 17 & 34 & 101 & 35 & 101 & 101 \\ 0 & 0 & 0 & 0 & 190 & 51 & 190 & 51 & 17 & 86 & 17 & 86 & 101 & 35 & 101 & 34 \end{array} \right]$$

The invariant subspace $\ker(M + Id)$ is generated by the following 12 row vectors, in the same representation.

$$\begin{aligned} &(1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\ &(0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\ &(0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 38 \ 0 \ 0 \ 159 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 38 \ 0 \ 0 \ 159 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 79 \ 0 \ 0 \ 38 \ 1) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 79 \ 0 \ 0 \ 38 \ 1) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0) \\ &(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1) \end{aligned}$$

B Well-Behaved Affine Spaces for the Robin and iSCREAM S-Box

Only spaces whose direction contains 1 are listed.

Values in $u + A$								Basis of A								Values in $v + B = S(u + A)$								Basis of B								dim($A + B$)	
00	01	26	27	84	85	a2	a3	01	26	84	00	01	26	27	84	85	a2	a3	01	26	84	01	26	84	01	26	84	3					
18	19	7c	7d	9e	9f	fa	fb	01	64	86	18	19	7c	7d	9e	9f	fa	fb	01	64	86	01	64	86	01	64	86	3					
28	29	32	33	8a	8b	90	91	01	1a	a2	90	91	8a	8b	32	33	28	29	01	1a	a2	01	1a	a2	01	1a	a2	3					
3c	3d	5e	5f	b2	b3	d0	d1	01	62	8e	b2	b3	d0	d1	3c	3d	5e	5f	01	62	8e	01	62	8e	01	62	8e	3					
44	45	66	67	c8	c9	ea	eb	01	22	8c	c8	c9	ea	eb	44	45	66	67	01	22	8c	01	22	8c	01	22	8c	3					
4e	4f	54	55	6c	6d	76	77	01	1a	22	77	76	6d	6c	55	54	4f	4e	01	1a	22	01	1a	22	01	1a	22	3					
28	29	32	33	6c	6d	76	77	01	1a	44	90	91	8a	8b	54	55	4e	4f	01	1a	c4	01	1a	c4	01	1a	c4	4					
28	29	32	33	4e	4f	54	55	01	1a	66	90	91	8a	8b	76	77	6c	6d	01	1a	e6	01	1a	e6	01	1a	e6	4					
2e	2f	38	39	8c	8d	9a	9b	01	16	a2	6f	6e	63	62	cd	cc	c1	c0	01	0c	a2	01	0c	a2	01	0c	a2	4					
08	09	2e	2f	8c	8d	aa	ab	01	26	84	4d	4c	6f	6e	c1	c0	e3	e2	01	22	8c	01	22	8c	01	22	8c	5					
08	09	38	39	9a	9b	aa	ab	01	30	92	4d	4c	63	62	cd	cc	e3	e2	01	2e	80	01	2e	80	01	2e	80	5					
0a	0b	12	13	c6	c7	de	df	01	18	cc	2c	2d	36	37	68	69	72	73	01	1a	44	01	1a	44	01	1a	44	5					
0e	0f	16	17	c2	c3	da	db	01	18	cc	bd	bc	ad	ac	f1	f0	e1	e0	01	10	4c	01	10	4c	01	10	4c	5					
20	21	3e	3f	86	87	98	99	01	1e	a6	59	58	5d	5c	e9	e8	ed	ec	01	04	b0	01	04	b0	01	04	b0	5					
22	23	34	35	80	81	96	97	01	16	a2	78	79	74	75	d8	d9	d4	d5	01	0c	a0	01	0c	a0	01	0c	a0	5					
24	25	3a	3b	82	83	9c	9d	01	1e	a6	47	46	43	42	fd	fc	f9	f8	01	04	ba	01	04	ba	01	04	ba	5					
4a	4b	50	51	8e	8f	94	95	01	1a	c4	e4	e5	f4	f5	a8	a9	b8	b9	01	10	4c	01	10	4c	01	10	4c	5					

References

1. CAESAR- Competition for Authenticated Encryption: Security, Applicability, and Robustness. General secretary Daniel J. Bernstein (2013). <http://competitions.cr.yp.to/caesar.html>
2. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçın, T.: Block ciphers – focus on the linear layer (feat. PRIDE). In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 57–76. Springer, Heidelberg (2014)
3. Bar-On, A., Dinur, I., Dunkelman, O., Lallemand, V., Tsaban, B.: Improved analysis of Zorro-like ciphers. Cryptology ePrint Archive, Report 2014/228 (2014). <http://eprint.iacr.org/>
4. Barkan, E., Biham, E.: In how many ways can you write rijndael? In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 160–175. Springer, Heidelberg (2002)
5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. IACR Cryptology ePrint Archive, 2013:414 (2013)
6. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
7. Borghoff, J., et al.: PRINCE – a low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
8. Bouillaguet, C., Dunkelman, O., Leurent, G., Fouque, P.-A.: Another look at complementation properties. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 347–364. Springer, Heidelberg (2010)
9. Bulygin, S., Walter, M., Buchmann, J.: Many weak keys for PRINTCIPHER: fast key recovery and countermeasures. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 189–206. Springer, Heidelberg (2013)
10. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
11. Chaum, D., Evertse, J.-H.: Cryptanalysis of des with a reduced number of rounds. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 192–211. Springer, Heidelberg (1986)
12. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: NESSIE proposal: NOEKEON (2000). <http://gro.noekeon.org/>
13. Evertse, J.-H.: Linear structures in block ciphers. In: Price, W.L., Chaum, D. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 249–266. Springer, Heidelberg (1988)
14. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.-X.: Block ciphers that are easier to mask: how far can we go? In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 383–399. Springer, Heidelberg (2013)
15. Grosso, V., Leurent, G., Standaert, F.-X., Varici, K.: LS-designs: bitslice encryption for efficient masked software implementations. To appear in the Proceedings of FSE 2014 (2014). <http://www.uclouvain.be/crypto/people/show/382>
16. Grosso, V., Leurent, G., Standaert, F.-X., Varici, K., Durvaux, F., Gaspar, L., Kerckhof, S.: Addendum to the CAESAR submission for SCREAM and iSCREAM. Posted on the official CAESAR submission list (2014). <http://competitions.cr.yp.to/round1/scream-ordering.txt>

17. Grosso, V., Leurent, G., Standaert, F.-X., Varici, K., Durvaux, F., Gaspar, L., Kerckhof, S.: CAESAR candidate SCREAM. Presentation by Gaëtan Leurent at DIAC 2014 (2014). <http://2014.diac.cr.yp.to/slides/leurent-scream.pdf>
18. Grosso, V., Leurent, G., Standaert, F.-X., Varici, K., Durvaux, F., Gaspar, L., Kerckhof, S.: SCREAM & iSCREAM. Entry in the CAESAR competition [1] (2014). <http://competitions.cr.yp.to/round1/screamv1.pdf>
19. Guo, J., Nikolić, I., Peyrin, T., Wang, L.: Cryptanalysis of Zorro. Cryptology ePrint Archive, Report 2013/713 (2013). <http://eprint.iacr.org/>
20. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Peneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
21. Karakoç, F., Demirci, H., Harmancı, A.E.: ITUbee: a software oriented lightweight block cipher. In: Avoine, G., Kara, O. (eds.) LightSec 2013. LNCS, vol. 8162, pp. 16–27. Springer, Heidelberg (2013)
22. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A cryptanalysis of PRINTCIPHER: the invariant subspace attack. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 206–221. Springer, Heidelberg (2011)
23. Leander, G., Minaud, B., Rønjom, S.: A generic approach to invariant subspace attacks: Cryptanalysis of Robin, iSCREAM and Zorro. Cryptology ePrint Archive, Report 2015/068 (2015). <http://eprint.iacr.org/2015/068>
24. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
25. Murphy, S.: An analysis of SAFER. Journal of Cryptology **11**(4), 235–251 (1998)
26. Prouff, E., Rivain, M.: Masking against side-channel attacks: a formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013)
27. Rasoolzadeh, S., Ahmadian, Z., Salmasizadeh, M., Aref, M.R.: Total break of Zorro using linear and differential attacks. Cryptology ePrint Archive, Report 2014/220 (2014). <http://eprint.iacr.org/>
28. Reeds, J.A., Manferdelli, J.L.: Des has no per round linear factors. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 377–389. Springer, Heidelberg (1985)
29. Sim, S.M., Wang, L.: Practical forgery attacks on SCREAM and iSCREAM (2014). Posted on the crypto competitions mailing list at <https://groups.google.com/d/forum/crypto-competitions>, report available at <https://www1.spms.ntu.edu.sg/~syllab/m/images/b/b3/ForgeryAttackOnSCREAM.pdf>
30. Soleimany, H.: Probabilistic slide cryptanalysis and its applications to LED-64 and Zorro. To appear in the Proceedings of FSE 2014 (2014). <http://research.ics.aalto.fi/publications/bibdb2014/pdf/fse2014.pdf>
31. Wang, Y., Wu, W., Guo, Z., Yu, X.: Differential cryptanalysis and linear distinguisher of full-round zorro. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 308–323. Springer, Heidelberg (2014)

Symmetric Cryptanalysis II

Structural Evaluation by Generalized Integral Property

Yosuke Todo^(✉)

NTT Secure Platform Laboratories, Tokyo, Japan
todo.yosuke@lab.ntt.co.jp

Abstract. In this paper, we show structural cryptanalyses against two popular networks, i.e., the Feistel Network and the Substitute-Permutation Network (SPN). Our cryptanalyses are distinguishing attacks by an improved integral distinguisher. The integral distinguisher is one of the most powerful attacks against block ciphers, and it is usually constructed by evaluating the propagation characteristic of integral properties, e.g., the ALL or BALANCE property. However, the integral property does not derive useful distinguishers against block ciphers with non-bijective functions and bit-oriented structures. Moreover, since the integral property does not clearly exploit the algebraic degree of block ciphers, it tends not to construct useful distinguishers against block ciphers with low-degree functions. In this paper, we propose a new property called *the division property*, which is the generalization of the integral property. It can effectively construct the integral distinguisher even if the block cipher has non-bijective functions, bit-oriented structures, and low-degree functions. From viewpoints of the attackable number of rounds or chosen plaintexts, the division property can construct better distinguishers than previous methods. Although our attack is a generic attack, it can improve several integral distinguishers against specific cryptographic primitives. For instance, it can reduce the required number of chosen plaintexts for the 10-round distinguisher on KECCAK-f from 2^{1025} to 2^{515} . For the Feistel cipher, it theoretically proves that SIMON 32, 48, 64, 96, and 128 have 9-, 11-, 11-, 13-, and 13-round integral distinguishers, respectively.

Keywords: Block cipher · Integral distinguisher · Feistel network · Substitute-Permutation network · KECCAK · SIMON · AES-like cipher · Boolean function

1 Introduction

The structural evaluation of cryptographic networks is an important topic of cryptology, and it helps a designer to design strong symmetric key primitives. There are several structural evaluations against the Feistel Network and the Substitute-Permutation Network (SPN) [6, 19, 22, 26, 28]. As one direction of the structural evaluation, there are the security evaluation by “the generic attack,”

Table 1. The number of required chosen plaintexts to construct r -round integral distinguishers on the SIMON family, Serpent, and KECCAK- f

Target	$\log_2(\# \text{texts})$								Method	Reference
	$r = 6$	$r = 7$	$r = 8$	$r = 9$	$r = 10$	$r = 11$	$r = 12$	$r = 13$		
SIMON 32	17	25	29	31	-	-	-	-	our	Sect. 4.3
	-	-	-	-	-	-	-	-	degree	[8, 21]
SIMON 48	17	29	39	44	46	47	-	-	our	Sect. 4.3
	17	-	-	-	-	-	-	-	degree	[8, 21]
SIMON 64	17	33	49	57	61	63	-	-	our	Sect. 4.3
	17	-	-	-	-	-	-	-	degree	[8, 21]
SIMON 96	17	33	57	77	87	92	94	95	our	Sect. 4.3
	17	33	-	-	-	-	-	-	degree	[8, 21]
SIMON 128	17	33	65	97	113	121	125	127	our	Sect. 4.3
	17	33	-	-	-	-	-	-	degree	[8, 21]
Target	$\log_2(\# \text{texts})$								Method	Reference
	$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$	$r = 9$	$r = 10$		
Serpent	12	28	84	113	124	-	-	-	our	Sect. 5.3
	28	82	113	123	127	-	-	-	degree	[9]
Target	$\log_2(\# \text{texts})$								Method	Reference
	$r = 8$	$r = 9$	$r = 10$	$r = 11$	$r = 12$	$r = 13$	$r = 14$	$r = 15$		
KECCAK- f	130	258	515	1025	1410	1538	1580	1595	our	Sect. 5.3
	257	513	1025	1409	1537	1579	1593	1598	degree	[9]

which exploits only the feature of the network and does not exploit the particular weaknesses of a specific cipher. It is applicable to large classes of block ciphers, but it is not often effective than the dedicated attack against the specific cipher. This paper focuses on generic attacks against both the Feistel Network and the SPN. The existing generic attack shows that the Feistel Network whose F -functions are chosen from random functions or permutations is vulnerable up to 5 rounds [22, 28]. Moreover, Biryukov and Shamir showed that the SPN is vulnerable up to 2.5 rounds [6].

Our Contribution. This paper shows generic attacks against two networks by improving an integral distinguisher. The integral attack was first proposed by Daemen et al. to evaluate the security of SQUARE [13], and then it was formalized by Knudsen and Wagner [23]. Nowadays, many integral distinguishers have been proposed against specific ciphers [23, 25, 35–37], and they are often constructed by evaluating the propagation characteristic of integral properties, e.g., the ALL property or the BALANCE property. In this paper, we revisit the integral property, and then introduce the *division property* by generalizing the integral property. The division property can effectively construct integral distinguishers even if block ciphers have non-bijective functions, bit-oriented structures, and low-degree functions.

The Feistel Network is a generic construction to create a (2ℓ) -bit pseudo-random permutation from an ℓ -bit pseudo-random function. We call the ℓ -bit function the F -function, and assume that an attacker can not know the specification of the F -function. Our distinguishing attack can attack up to 3 rounds, and it can attack up to 5 rounds if the F -function is limited to a permutation. Unfortunately, they are not improved compared with the previous ones. However, assuming that the algebraic degree of the F -function is smaller than the bit length of the F -function, our attack can attack more rounds than the previous attacks exploiting the low-degree function. We summarize new integral distinguishers in Appendix B. Although the assumption of our attack is only the algebraic degree of the F -function, it can construct new integral distinguishers on the SIMON family [5]. Since SIMON has a non-bijective F -function and a bit-oriented structure, it is complicated task to construct the integral distinguisher. The division property theoretically introduces that SIMON 32, 48, 64, 96, and 128 have at least 9-, 11-, 11-, 13-, and 13-round integral distinguishers, respectively. Table 1 shows the comparison between our distinguishers and previous ones.

The SPN consists of an S-Layer and a P-Layer, where the S-Layer has m ℓ -bit bijective S-boxes and the P-Layer has an (ℓm) -bit bijective linear function. The attacker can not know the specifications of the S-boxes and the linear function. Surprisingly, our generic attack becomes able to attack more rounds as the number of S-boxes is larger than the bit length of the S-box. This fact implies that the design of the P-Layer that can diffuse more outputs of S-boxes may not derive prospective security improvements. We summarize new integral distinguishers in Appendix C. Similar to the result against the Feistel Network, the division property is also useful to construct integral distinguishers against specific cryptographic primitives. For instance, we can reduce the required number of chosen plaintexts for the 7-round distinguisher on Serpent [1] from 2^{127} to 2^{124} . Moreover, for the integral distinguisher on KECCAK- f [12], we can reduce the required number of chosen plaintexts compared with previous ones constructed by Boura et al. [9]. Table 1 shows the comparison between our distinguishers and previous ones.

Organization. This paper is organized as follows: In Sect. 2, we show notations, Boolean functions, and the framework of integral distinguishers. In Sect. 3, we propose the division property by generalizing the integral property, and show the propagation characteristic. In Sect. 4 and Sect. 5, we show new distinguishing attacks on the Feistel Network and the SPN, respectively. In Sect. 6, we show that the division property is also useful to construct the dedicated attack against specific ciphers. As an example, we show new distinguishing attacks on the AES-like cipher. Section 7 concludes this paper.

2 Preliminaries

2.1 Notation

We make the distinction between addition of \mathbb{F}_2^n and addition of \mathbb{Z} , and we use \oplus and $+$ as addition of \mathbb{F}_2^n and addition of \mathbb{Z} , respectively. For any $a \in \mathbb{F}_2^n$, the

i -th element is expressed in $a[i]$ and the hamming weight w_a is calculated as $w_a = \sum_{i=1}^n a[i]$. Let $1^n \in \mathbb{F}_2^n$ be a value whose all elements are 1. Moreover, let $0^n \in \mathbb{F}_2^n$ be a value whose all elements are 0.

Subsets \mathbb{S}_k^n and $\mathbb{S}_{\mathbf{k}}^{n,m}$. Let \mathbb{S}_k^n be a subset of \mathbb{F}_2^n for any integer $k \in \{0, 1, \dots, n\}$. The subset \mathbb{S}_k^n is a set of all $a \in \mathbb{F}_2^n$ satisfying $k \leq w_a$, and it is defined as

$$\mathbb{S}_k^n := \{a \in \mathbb{F}_2^n \mid k \leq w_a\}.$$

Let $\mathbb{S}_{\mathbf{k}}^{n,m}$ be a subset of $(\mathbb{F}_2^n)^m$ for any vector $\mathbf{k} \in (\{0, 1, \dots, n\})^m$. The subset $\mathbb{S}_{\mathbf{k}}^{n,m}$ is a set of all $\mathbf{a} \in (\mathbb{F}_2^n)^m$ satisfying $k_i \leq w_{a_i}$, and it is defined as

$$\mathbb{S}_{\mathbf{k}}^{n,m} := \{(a_1, a_2, \dots, a_m) \in (\mathbb{F}_2^n)^m \mid k_i \leq w_{a_i} \text{ for } 1 \leq i \leq m\}.$$

Bit Product Functions π_u and $\pi_{\mathbf{u}}$. Let $\pi_u : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ be a function for any $u \in \mathbb{F}_2^n$. Let $x \in \mathbb{F}_2^n$ be an input of π_u , and $\pi_u(x)$ is the AND of $x[i]$ satisfying $u[i] = 1$, namely, it is defined as

$$\pi_u(x) := \prod_{i=1}^n x[i]^{u[i]}.$$

Let $\pi_{\mathbf{u}} : (\mathbb{F}_2^n)^m \rightarrow \mathbb{F}_2$ be a function for any $\mathbf{u} \in (\mathbb{F}_2^n)^m$. Let $\mathbf{x} \in (\mathbb{F}_2^n)^m$ be an input of $\pi_{\mathbf{u}}$, namely, $\pi_{\mathbf{u}}(\mathbf{x})$ is calculated as

$$\pi_{\mathbf{u}}(\mathbf{x}) := \prod_{i=1}^m \pi_{u_i}(x_i).$$

2.2 Boolean Function

A Boolean function is a function from \mathbb{F}_2^n to \mathbb{F}_2 . Let $\deg(f)$ be the algebraic degree of a Boolean function f . As representations of the Boolean function, we use Algebraic Normal Form, which is defined as follows.

Algebraic Normal Form. Algebraic Normal Form (ANF) is a representation of a Boolean function. Any $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ can be represented as

$$f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u^f \left(\prod_{i=1}^n x[i]^{u[i]} \right) = \bigoplus_{u \in \mathbb{F}_2^n} a_u^f \pi_u(x),$$

where $a_u^f \in \mathbb{F}_2$ is a constant value depending on f and u . If $\deg(f)$ is at most d , all a_u^f satisfying $d < w_u$ are 0. An n -bit S-box can be regarded as the concatenation of n Boolean functions. If algebraic degrees of n Boolean functions are at most d , we say the algebraic degree of the S-box is at most d .

$\begin{array}{ c c c c } \hline A & c & c & c \\ \hline c & A & c & c \\ \hline c & c & A & c \\ \hline c & c & c & A \\ \hline \end{array}$	$\begin{array}{ c c c c } \hline A & c & c & c \\ \hline c & C & C & C \\ \hline C & C & C & C \\ \hline C & C & C & C \\ \hline \end{array}$	$\begin{array}{ c c c c } \hline A & c & c & c \\ \hline A & c & c & c \\ \hline A & C & C & C \\ \hline A & C & C & C \\ \hline \end{array}$	$\begin{array}{ c c c c } \hline A & A & A & A \\ \hline A & A & A & A \\ \hline A & A & A & A \\ \hline A & A & A & A \\ \hline \end{array}$	$\begin{array}{ c c c c } \hline B & B & B & B \\ \hline B & B & B & B \\ \hline B & B & B & B \\ \hline B & B & B & B \\ \hline \end{array}$
SB SR MC AK				
2^{24} sets				

Fig. 1. Integral distinguisher on 4-round AES

2.3 Integral Distinguisher

An integral distinguisher was first proposed by Daemen et al. to evaluate the security of SQUARE [13], and then it was formalized by Knudsen and Wagner [23]. It uses a set of chosen plaintexts that contains all possible values for some bits and has a constant value for the other bits. Corresponding ciphertexts are calculated from plaintexts in the set by using an encryption oracle. If the XOR of the corresponding ciphertexts always becomes 0, we say that this cipher has the integral distinguisher.

Integral Property. Nowadays, many integral distinguishers have been proposed against specific ciphers [23, 25, 35–37], and they are often constructed by evaluating the propagation characteristic of the integral property. We define four integral properties as follows:

- ALL (\mathcal{A}) : Every value appears the same number in the multiset.
- BALANCE (\mathcal{B}) : The XOR of all texts in the multiset is 0.
- CONSTANT (\mathcal{C}) : The value is fixed to a constant for all texts in the multiset.
- UNKNOWN (\mathcal{U}) : The multiset is indistinguishable from one of n -bit random values.

Knudsen and Wagner showed that AES has the 4-round integral distinguisher with 2^{32} chosen plaintexts [23]. Figure 1 shows the integral distinguisher.

Unfortunately, the integral property does not derive effective distinguishers if block ciphers consist of non-bijective functions, e.g., DES [31] and SIMON [5] consist of non-bijection functions. Moreover, since the propagation characteristic does not clearly exploit the algebraic degree of block ciphers, it tends not to construct effective distinguishers against block ciphers with low-degree round functions.

Degree Estimation. As another method to construct the integral distinguisher, there is a higher-order differential attack [21, 24], which exploits the algebraic degree of block ciphers. When the algebraic degree of a block cipher is at most D , the cipher has the integral distinguisher with 2^{D+1} chosen plaintexts. Canteaut and Videau showed the bound of the degree of iterated round functions [11]. Then, Boura et al. improved the bound [9], and showed integral distinguishers on KECCAK [12] and *Luffa* [10]. We show the bound in Appendix A.

3 Division Property

3.1 Introduction of Division Property

We propose a new property called *the division property*, which is the generalization of the integral property. We consider one bijective S-box with degree d . If an input multiset has \mathcal{A} , the output multiset also has \mathcal{A} . If an input multiset has \mathcal{B} , the output multiset has \mathcal{U} . If we have the input multiset with 2^{d+1} chosen texts, the output multiset has \mathcal{B} because the degree of the S-box is d . The integral property does not exploit this property. We now want to exploit useful properties that are hidden between \mathcal{A} and \mathcal{B} . Therefore, we redefine \mathcal{A} and \mathcal{B} by the same notation, and then introduce the division property by generalizing the redefinition.

Redefinition of Integral Property. Let \mathbb{X} be a multiset whose elements take an n -bit value. We first consider features of the multiset \mathbb{X} satisfying \mathcal{A} . If we choose one bit from n bits and calculate the XOR of the chosen bit in the multiset, the calculated value is always 0. Moreover, if we choose at most $(n - 1)$ bits from n bits and calculate the XOR of the AND of chosen bits in the multiset, the calculated value is also always 0. However, if we choose all bits from n bits and calculate the XOR of the AND of n bits in the multiset, the calculated value becomes unknown¹. Above features are expressed by using the bit product function π_u , which is defined in Sect. 2.1, as follows. We evaluate the parity of $\pi_u(x)$ for all $x \in \mathbb{X}$, namely, evaluate $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$. The parity is always even for any u satisfying $w_u < n$. On the other hand, the parity becomes unknown for $u = 1^n$.

We next consider features of the multiset \mathbb{X} satisfying \mathcal{B} . If we choose one bit from n bits and calculate the XOR of the chosen bit in the multiset, the calculated value is always 0. However, if we choose at least two bits from n bits and calculate the XOR of the AND of chosen bits in the multiset, the calculated value becomes unknown. Above features are expressed by using the bit product function π_u as follows. We evaluate the parity of $\pi_u(x)$ for all $x \in \mathbb{X}$. The parity is always even for any u satisfying $w_u < 2$. On the other hand, the parity becomes unknown for any u satisfying $w_u \geq 2$.

3.2 Definition of Division Property

Section 3.1 redefines both the ALL and BALANCE properties by the same notation. Since the redefinition can be parameterized by the number of product bits w_u of the bit product function π_u , we generalize the integral property as follows.

¹ If all values appear the same even number in the multiset, the calculated value is always 0. If all values appear the same odd number in the multiset, the calculated value is always 1. Thus, we cannot guarantee whether the calculated value is 0 or not when we consider the multiset satisfying \mathcal{A} . In this case, we say the calculated value becomes unknown.

Definition 1 (Division Property). Let \mathbb{X} be a multiset whose elements take a value of \mathbb{F}_2^n , and k takes a value between 0 and n . When the multiset \mathbb{X} has the division property \mathcal{D}_k^n , it fulfills the following conditions: The parity of $\pi_u(x)$ for all $x \in \mathbb{X}$ is always even if w_u is less than k . Moreover, the parity becomes unknown if w_u is greater than or equal to k .

When the multiset \mathbb{X} has \mathcal{D}_k^n , it satisfies

$$\bigoplus_{x \in \mathbb{X}} \pi_u(x) = 0, \text{ for all } u \in (\mathbb{F}_2^n \setminus \mathbb{S}_k^n),$$

where \mathbb{S}_k^n is a subset defined in Sect. 2.1. The parity of $\pi_u(x)$ for all $x \in \mathbb{X}$ becomes unknown for any $u \in \mathbb{S}_k^n$. Namely, in the division property, the set of u is divided into the subset that $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ becomes unknown and the subset that $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ becomes 0.

Example 1. Let \mathbb{X} be a multiset whose elements take a value of \mathbb{F}_2^4 . As an example, we prepare the input multiset \mathbb{X} as

$$\mathbb{X} := \{0x0, 0x3, 0x3, 0x3, 0x5, 0x6, 0x8, 0xB, 0xD, 0xE\}.$$

A following table calculates the summation of $\pi_u(x)$.

	0x0	0x3	0x3	0x3	0x5	0x6	0x8	0xB	0xD	0xE	$\sum \pi_u(x)$ $(\bigoplus \pi_u(x))$
	0000	0011	0011	0011	0101	0110	1000	1011	1101	1110	
$u = 0000$	1	1	1	1	1	1	1	1	1	1	10 (0)
$u = 0001$	0	1	1	1	1	0	0	1	1	0	6 (0)
$u = 0010$	0	1	1	1	0	1	0	1	0	1	6 (0)
$u = 0011$	0	1	1	1	0	0	0	1	0	0	4 (0)
$u = 0100$	0	0	0	0	1	1	0	0	1	1	4 (0)
$u = 0101$	0	0	0	0	1	0	0	0	1	0	2 (0)
$u = 0110$	0	0	0	0	0	1	0	0	0	1	2 (0)
$u = 0111$	0	0	0	0	0	0	0	0	0	0	0 (0)
$u = 1000$	0	0	0	0	0	0	1	1	1	1	4 (0)
$u = 1001$	0	0	0	0	0	0	0	1	1	0	2 (0)
$u = 1010$	0	0	0	0	0	0	0	1	0	1	2 (0)
$u = 1011$	0	0	0	0	0	0	0	1	0	0	1 (1)
$u = 1100$	0	0	0	0	0	0	0	0	1	1	2 (0)
$u = 1101$	0	0	0	0	0	0	0	0	1	0	1 (1)
$u = 1110$	0	0	0	0	0	0	0	0	0	1	1 (1)
$u = 1111$	0	0	0	0	0	0	0	0	0	0	0 (0)

For all u satisfying $w_u < 3$, $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ becomes 0. Therefore, the multiset has the division property \mathcal{D}_3^4 .

Each definition of \mathcal{B} and \mathcal{U} is essentially the same as that of \mathcal{D}_2^n and \mathcal{D}_1^n , respectively. However, the definition of \mathcal{A} is different from that of \mathcal{D}_n^n . The multiset satisfying \mathcal{A} always has the division property \mathcal{D}_n^n but not vice versa. For instance, the multiset satisfying the EVEN property, which is defined that the number of occurrences is even for all values [30], does not always have \mathcal{A} , but it always has \mathcal{D}_n^n . In this paper, we use only \mathcal{D}_n^n instead of \mathcal{A} because it is sufficient to use \mathcal{D}_n^n from the viewpoint of the construction of integral distinguishers.

Propagation Characteristic of Division Property Let s be an S-box whose degree is d . Let \mathbb{X} be an input multiset whose elements take a value of \mathbb{F}_2^n . Let \mathbb{Y} be an output multiset whose elements are calculated from $s(x)$ for all $x \in \mathbb{X}$. We assume that \mathbb{X} has \mathcal{D}_k^n , and want to evaluate the division property of \mathbb{Y} . In the division property, the set of u is divided into the subset that $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ becomes unknown and the subset that $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ becomes 0. Therefore, we divide the set of v into the subset that $\bigoplus_{s(x) \in \mathbb{Y}} \pi_v(s(x))$ becomes unknown and the subset that $\bigoplus_{s(x) \in \mathbb{Y}} \pi_v(s(x))$ becomes 0. Since the parity of $\pi_v(s(x))$ for all $s(x) \in \mathbb{Y}$ is equal to that of $(\pi_v \circ s)(x)$ for all $x \in \mathbb{X}$, we evaluate $\bigoplus_{x \in \mathbb{X}} (\pi_v \circ s)(x)$.

Proposition 1 (Propagation Characteristic of Division Property). *Let s be an function (S-box) from n bits to n bits, and the degree is d . Assuming that an input multiset \mathbb{X} has the division property \mathcal{D}_k^n , the output multiset \mathbb{Y} has $\mathcal{D}_{\lceil \frac{k}{d} \rceil}^n$. In addition, assuming that the S-box is a permutation, the output multiset \mathbb{Y} has \mathcal{D}_n^n when the input multiset has \mathcal{D}_n^n .*

Proof. We represent $\bigoplus_{x \in \mathbb{X}} (\pi_v \circ s)(x)$ by using ANF as

$$\begin{aligned} \bigoplus_{x \in \mathbb{X}} (\pi_v \circ s)(x) &= \bigoplus_{x \in \mathbb{X}} \left(\bigoplus_{u \in \mathbb{F}_2^n} a_u^{\pi_v \circ s} \pi_u(x) \right) \\ &= \bigoplus_{u \in \mathbb{S}_k^n} a_u^{\pi_v \circ s} \left(\bigoplus_{x \in \mathbb{X}} \pi_u(x) \right) \oplus \bigoplus_{u \in (\mathbb{F}_2^n \setminus \mathbb{S}_k^n)} a_u^{\pi_v \circ s} \left(\bigoplus_{x \in \mathbb{X}} \pi_u(x) \right). \end{aligned}$$

Since the multiset \mathbb{X} has \mathcal{D}_k^n , $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ is always 0 for any $u \in (\mathbb{F}_2^n \setminus \mathbb{S}_k^n)$. Therefore, it satisfies

$$\bigoplus_{x \in \mathbb{X}} (\pi_v \circ s)(x) = \bigoplus_{u \in \mathbb{S}_k^n} a_u^{\pi_v \circ s} \left(\bigoplus_{x \in \mathbb{X}} \pi_u(x) \right).$$

If $a_u^{\pi_v \circ s}$ is 0 for all $u \in \mathbb{S}_k^n$, $\bigoplus_{x \in \mathbb{A}} (\pi_v \circ s)(x)$ always becomes 0. In other words, if there exists $u \in \mathbb{S}_k^n$ such that $a_u^{\pi_v \circ s}$ is 1, $\bigoplus_{x \in \mathbb{A}} (\pi_v \circ s)(x)$ becomes unknown. Since the function π_v is the AND of w_v bits and the degree of S-box is d , the degree of the Boolean function $(\pi_v \circ s)$ has the following properties:

- The degree of $(\pi_v \circ s)$ is at most $\min\{n - 1, w_v \times d\}$.
- If the S-box is a permutation, the degree of $(\pi_v \circ s)$ is at most $n - 1$ for $w_v < n$.

We first assume that the multiset \mathbb{X} has \mathcal{D}_k^n . In this case, we consider only u satisfying $w_u \geq k$. When $w_v \times d < k$ holds, $a_u^{\pi_v \circ s}$ is always 0. Thus, the necessary condition that $a_u^{\pi_v \circ s}$ becomes 1 is $w_v \times d \geq k$, and it is $w_v \geq \lceil \frac{k}{d} \rceil$. Namely, the necessary condition that $\bigoplus_{x \in \mathbb{X}} (\pi_v \circ s)(x)$ becomes unknown is $w_v \geq \lceil \frac{k}{d} \rceil$, and \mathbb{Y} has $\mathcal{D}_{\lceil \frac{k}{d} \rceil}^n$. We next assume that the multiset \mathbb{X} has \mathcal{D}_n^n and the S-box is a permutation. In this case, we consider only $u = 1^n$. When $w_v < n$ holds, $a_{1^n}^{\pi_v \circ s}$

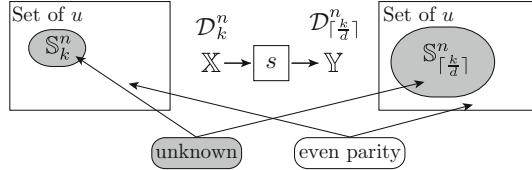


Fig. 2. Propagation characteristic of division property

is always 0 because the degree of the Boolean function $(\pi_v \circ s)$ is at most $n - 1$. Thus, the necessary condition that $a_{1^n}^{\pi_v \circ s}$ becomes 1 is $v = 1^n$. Namely, the necessary condition that $\bigoplus_{x \in \mathbb{X}} (\pi_v \circ s)(x)$ becomes unknown is $v = 1^n$, and \mathbb{Y} has \mathcal{D}_n^n . \square

Example 2. Let us consider a following 4-bit S-box.

x	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
$s(x)$	0x8	0xC	0x0	0xB	0x9	0xD	0xE	0x5	0xA	0x1	0x2	0x6	0x4	0xF	0x3	0x7

The S-box is bijective and the algebraic degree is 2. We now prepare the input multiset $\mathbb{X} := \{0x0, 0x3, 0x3, 0x3, 0x5, 0x6, 0x8, 0xB, 0xD, 0xE\}$, which is the same as Example 1 and the division property is \mathcal{D}_3^4 . The output multiset is calculated as $\mathbb{Y} := \{0x8, 0xB, 0xB, 0xB, 0xD, 0xE, 0xA, 0x6, 0xF, 0x3\}$, and a following table calculates the summation of $\pi_v(y)$.

	0x8	0xB	0xB	0xB	0xD	0xE	0xA	0x6	0xF	0x3	$\sum \pi_v(y)$ $(\bigoplus \pi_v(y))$
	1000	1011	1011	1011	1101	1110	1010	0110	1111	0011	
$v = 0000$	1	1	1	1	1	1	1	1	1	1	10 (0)
$v = 0001$	0	1	1	1	1	0	0	0	1	1	6 (0)
$v = 0010$	0	1	1	1	0	1	1	1	1	1	8 (0)
$v = 0011$	0	1	1	1	0	0	0	0	1	1	5 (1)
$v = 0100$	0	0	0	0	1	1	0	1	1	0	4 (0)
$v = 0101$	0	0	0	0	1	0	0	0	1	0	2 (0)
$v = 0110$	0	0	0	0	0	1	0	1	1	0	3 (1)
$v = 0111$	0	0	0	0	0	0	0	0	1	0	1 (1)
$v = 1000$	1	1	1	1	1	1	0	1	0	8 (0)	
$v = 1001$	0	1	1	1	1	0	0	0	1	0	5 (1)
$v = 1010$	0	1	1	1	0	1	1	0	1	0	6 (0)
$v = 1011$	0	1	1	1	0	0	0	0	1	0	4 (0)
$v = 1100$	0	0	0	0	1	1	0	0	1	0	3 (1)
$v = 1101$	0	0	0	0	1	0	0	0	1	0	2 (0)
$v = 1110$	0	0	0	0	0	1	0	0	1	0	2 (0)
$v = 1111$	0	0	0	0	0	0	0	0	1	0	1 (1)

For all v satisfying $w_v < 2$, $\bigoplus_{y \in \mathbb{Y}} \pi_v(y)$ becomes 0. Therefore, the multiset \mathbb{Y} has the division property \mathcal{D}_2^4 .

Figure 2 shows the outline of the propagation characteristic of the division property. Let \mathbb{X} and \mathbb{Y} be input and output multisets, respectively. First, the

size of the set of u that $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ becomes unknown is small. However, the size of the set of u that $\bigoplus_{x \in \mathbb{X}} \pi_u(s(x))$ becomes unknown expands. If the size expands to the universal set except for 0^n , we regard that the output multiset is indistinguishable from the multiset of random texts.

3.3 Vectorial Division Property

Section 3.2 only shows the division property for one S-box. However, since practical ciphers use several S-boxes in every round, we can not construct integral distinguishers by only using Proposition 1. Therefore, we vectorize the division property.

Let an S-Layer be any function that consists of m n -bit S-boxes with degree d in parallel. We now consider the propagation characteristic of the division property against the S-Layer. Let \mathbb{X} be the input multiset of the S-Layer, and $x \in \mathbb{X}$ takes a value of $(\mathbb{F}_2^n)^m$. The vectorization is the natural extension of the division property. Namely, the set of u is divided into the subset that $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ becomes unknown and the subset that $\bigoplus_{x \in \mathbb{X}} \pi_u(x)$ becomes 0, where u is an m -dimensional vector whose elements take a value of \mathbb{F}_2^n . Figure 3 shows the difference between the division property and the vectorial one.

Definition 2 (Vectorial Division Property). *Let \mathbb{X} be the multiset whose elements take a value of $(\mathbb{F}_2^n)^m$, and k is an m -dimensional vector whose elements take a value between 0 and n . When the multiset \mathbb{X} has the division property $D_k^{n,m}$, the multiset fulfills the following conditions: The parity of $\pi_u(x)$ for all $x \in \mathbb{X}$ is always even if u does NOT belong to $S_k^{n,m}$. Moreover, the parity becomes unknown if u belongs to $S_k^{n,m}$.*

Propagation Characteristic of Vectorial Division Property. Assume that the input multiset of the S-Layer has the division property $D_k^{n,m}$. The output of the S-Layer is calculated as $S(x) = (s_1(x_1), s_2(x_2), \dots, s_m(x_m))$ for $(x_1, x_2, \dots, x_m) \in \mathbb{X}$. We now consider the set of v that $\bigoplus_{x \in \mathbb{X}} \pi_v(S(x))$ becomes unknown and the set of v that $\bigoplus_{x \in \mathbb{X}} \pi_v(S(x))$ becomes 0. Since the output of each S-box is calculated independently, the propagation characteristic of the division property can also be evaluated independently. Namely, the output multiset has $D_{k'}^{n,m}$, where $k'_i = \lceil k_i/d \rceil$ holds. Moreover, if the S-box is bijective and $k_i = n$ holds, $k'_i = n$ holds.

3.4 Collective Division Property

By vectorizing of the division property, we can evaluate the multiset whose elements take a value of $(\mathbb{F}_2^n)^m$. However, it is still insufficient to use only vectorial division property. For simplicity, we consider a multiset \mathbb{X} whose elements take a value of $(\mathbb{F}_2^8)^2$. Assume that the number of elements in \mathbb{X} is 256, and two elements of x take all values from 0 to 255 independently. We consider the set of u that the parity of $\pi_u(x)$ for all $x \in \mathbb{X}$ becomes unknown and the set of u that the parity becomes 0.

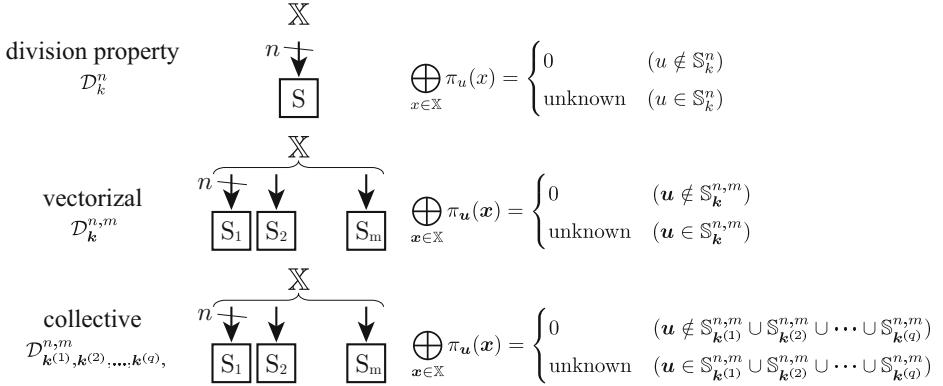


Fig. 3. Division property, vectorial division property, and collective division property

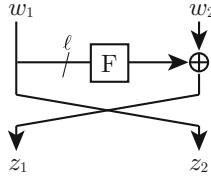
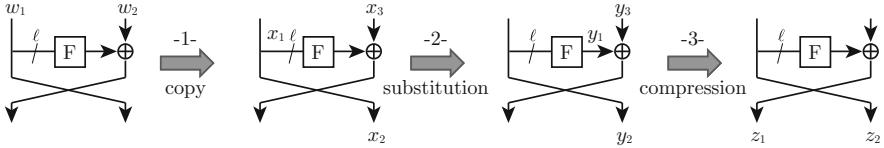
- The parity becomes unknown if u belongs to $\mathbb{S}_{[8,0]}^{8,2}$.
- The parity becomes unknown if u belongs to $\mathbb{S}_{[0,8]}^{8,2}$.
- The parity becomes unknown if u belongs to $\mathbb{S}_{[1,1]}^{8,2}$.
- Otherwise, i.e., u does NOT belong to $\mathbb{S}_{[8,0]}^{8,2} \cup \mathbb{S}_{[0,8]}^{8,2} \cup \mathbb{S}_{[1,1]}^{8,2}$, the parity is always even.

We can not express this property by using the vectorial division property. Therefore, we collect several vectorial division properties. Figure 3 shows the difference between the vectorial division property and the collective division property.

Definition 3 (Collective Division Property). Let \mathbb{X} be the multiset whose elements take a value of $(\mathbb{F}_2^n)^m$, and $\mathbf{k}^{(j)}$ ($j = 1, 2, \dots, q$) are m -dimensional vectors whose elements take a value between 0 and n . When the multiset \mathbb{X} has the division property $\mathcal{D}_{\mathbf{k}^{(1)}, \mathbf{k}^{(2)}, \dots, \mathbf{k}^{(q)}}^{n,m}$, the multiset fulfills the following conditions: The parity of $\pi_u(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{X}$ is always even if u does NOT belong to the union $\mathbb{S}_{\mathbf{k}^{(1)}}^{n,m} \cup \mathbb{S}_{\mathbf{k}^{(2)}}^{n,m} \cup \dots \cup \mathbb{S}_{\mathbf{k}^{(q)}}^{n,m}$. Moreover, the parity becomes unknown if u belongs to the union $\mathbb{S}_{\mathbf{k}^{(1)}}^{n,m} \cup \mathbb{S}_{\mathbf{k}^{(2)}}^{n,m} \cup \dots \cup \mathbb{S}_{\mathbf{k}^{(q)}}^{n,m}$.

It is obvious that the collective division property with $q = 1$ is the same as the vectorial division property.

Propagation Characteristic of Collective Division Property. Assume that the input multiset of the S-Layer has the division property $\mathcal{D}_{\mathbf{k}^{(1)}, \mathbf{k}^{(2)}, \dots, \mathbf{k}^{(q)}}^{n,m}$. We now consider the set of v that $\bigoplus_{\mathbf{x} \in \mathbb{X}} \pi_v(S(\mathbf{x}))$ becomes unknown, and the set is derived from only the set of u that $\bigoplus_{\mathbf{x} \in \mathbb{X}} \pi_u(\mathbf{x})$ becomes unknown. Therefore, we can evaluate the propagation characteristic of $\mathcal{D}_{\mathbf{k}^{(j)}}^{n,m}$ for all j independently. Namely, the output multiset has $\mathcal{D}_{\mathbf{k}'^{(1)}, \mathbf{k}'^{(2)}, \dots, \mathbf{k}'^{(q)}}^{n,m}$, where $k_i'^{(j)} = \lceil k_i^{(j)} / d \rceil$ holds. Moreover, if the S-box is bijective and $k_i^{(j)} = n$ holds, $k_i'^{(j)} = n$ holds.

**Fig. 4.** (ℓ, d) -Feistel**Fig. 5.** Propagation characteristic for Feistel Network

4 Improved Integral Distinguishers on Feistel Network

4.1 Feistel Network

(ℓ, d) -Feistel. The Feistel Network is one of the most popular network to design block ciphers. When n -bit block ciphers are constructed by the Feistel Network, the input of the round function is expressed in two $(n/2)$ -bit values. Moreover, an $(n/2)$ -bit non-linear function F is used in the round function, and we call this function the F -function. Let (w_1, w_2) be the input of the round function, and the output is calculated as $(z_1, z_2) = (F(w_1) \oplus w_2, w_1)$. We now define an (ℓ, d) -Feistel, whose F -function is an ℓ -bit non-linear function with degree d (this function is not limited to a permutation). Figure 4 shows the round function of the Feistel Network. There are many block ciphers adopting (ℓ, d) -Feistel, e.g. DES [31], Camellia [3], and SIMON 2n [5] adopt $(32, 5)$ -, $(64, 7)$ -, and $(n, 2)$ -Feistel, respectively.

4.2 Propagation Characteristic for Feistel Network

This section shows that the division property is useful to construct integral distinguishers on (ℓ, d) -Feistel. Since the Feistel Network has “copy,” “substitution,” and “compression,” we need to propagate the division property against them. The “copy” creates the input of the F -function, and the “substitution” processes the input by the F -function, and finally the “compression” creates the left half of the output by XOR. Figure 5 shows the outline of the propagation characteristic.

-1- Copy. Let \mathbb{W} be an input set, and $(w_1, w_2) \in \mathbb{W}$ denotes the input value.

The round function first creates (x_1, x_2, x_3) , where $x_1 = w_1$, $x_2 = w_1$, and

$x_3 = w_2$ hold. Here, x_1 is the input of the F -function, x_2 is the right half of the output of the round function, and x_3 is the right half of the input of the round function. Let \mathbb{X} be the output set whose elements take (x_1, x_2, x_3) for all $(w_1, w_2) \in \mathbb{W}$. Assume that the input set \mathbb{W} has the division property $\mathcal{D}_{\mathbf{k}^{(1)}, \mathbf{k}^{(2)}, \dots, \mathbf{k}^{(q)}}^{\ell, 2}$. If we use $\pi_{\mathbf{u}}$ satisfying $k_1^{(j)} \leq u_1$ and $k_2^{(j)} \leq u_2$, the parity of $\pi_{\mathbf{u}}(\mathbf{w})$ for all $\mathbf{w} \in \mathbb{W}$ becomes unknown. Since x_1 is equal to x_2 , the parity of $\pi_{\mathbf{v}}(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{X}$ becomes unknown if we use $\pi_{\mathbf{v}}$ satisfying $k_1^{(j)} - k' \leq v_1$, $k' \leq v_2$, and $k_2^{(j)} \leq v_3$ for all $k' (0 \leq k' \leq k_1^{(j)})$. Therefore, the set \mathbb{X} has the division property $\mathcal{D}_{[0, k_1^{(1)}, k_2^{(1)}], [1, k_1^{(1)} - 1, k_2^{(1)}], \dots, [k_1^{(1)}, 0, k_2^{(1)}], \dots, [k_1^{(q)}, 0, k_2^{(q)}]}^{\ell, 3}$.

-2- Substitution. The F -function is an ℓ -bit function with degree d . Assume that the input set has the division property $\mathcal{D}_{\mathbf{k}^{(1)}, \mathbf{k}^{(2)}, \dots, \mathbf{k}^{(q)}}^{\ell, 3}$. From the propagation characteristic of the division property, the output set has $\mathcal{D}_{\mathbf{k}'^{(1)}, \mathbf{k}'^{(2)}, \dots, \mathbf{k}'^{(q)}}^{\ell, 3}$, where $(k_1'^{(j)}, k_2'^{(j)}, k_3'^{(j)}) = ([k_1^{(j)}/d], k_2^{(j)}, k_3^{(j)})$ holds. If the F -function is limited to a permutation, $k_1'^{(j)}$ becomes ℓ when $k_1^{(j)} = \ell$ holds.

-3- Compression. Let \mathbb{Y} be the input set, and $(y_1, y_2, y_3) \in \mathbb{Y}$ denotes the input value, where y_1 denotes the output of the F -function. Let y_1 be XORed with y_3 , and then the internal state is expressed in $(z_1, z_2) = (y_1 \oplus y_3, y_2)$. Let \mathbb{Z} be the set whose elements take (z_1, z_2) for all $(y_1, y_2, y_3) \in \mathbb{Y}$. To evaluate the division property of the set \mathbb{Z} , we calculate the parity of $\pi_{\mathbf{v}}(z_1, z_2)$ for all $(z_1, z_2) \in \mathbb{Z}$ as

$$\begin{aligned} \bigoplus_{(z_1, z_2) \in \mathbb{Z}} \pi_{\mathbf{v}}(z_1, z_2) &= \bigoplus_{(z_1, z_2) \in \mathbb{Z}} (\pi_{v_1}(z_1) \times \pi_{v_2}(z_2)) \\ &= \bigoplus_{(y_1, y_2, y_3) \in \mathbb{Y}} (\pi_{v_1}(y_1 \oplus y_3) \times \pi_{v_2}(y_2)) \\ &= \bigoplus_{(y_1, y_2, y_3) \in \mathbb{Y}} \left(\bigoplus_{c \preceq v_1} (\pi_c(y_1) \times \pi_{v_1 \oplus c}(y_3)) \times \pi_{v_2}(y_2) \right) \\ &= \bigoplus_{c \preceq v_1} \left(\bigoplus_{(y_1, y_2, y_3) \in \mathbb{Y}} \pi_c(y_1) \times \pi_{v_2}(y_2) \times \pi_{v_1 \oplus c}(y_3) \right), \end{aligned}$$

where the set of c chosen from $c \preceq v_1$ denotes the set of c satisfying $c \wedge v_1 = c$. Assuming that the input set \mathbb{Y} has the division property $\mathcal{D}_{\mathbf{k}^{(1)}, \mathbf{k}^{(2)}, \dots, \mathbf{k}^{(q)}}^{\ell, 3}$, the output set \mathbb{Z} has the division property $\mathcal{D}_{\mathbf{k}'^{(1)}, \mathbf{k}'^{(2)}, \dots, \mathbf{k}'^{(q)}}^{\ell, 2}$, where $(k_1'^{(j)}, k_2'^{(j)}) = (k_1^{(j)} + k_3^{(j)}, k_2^{(j)})$ holds. Notice that the parity of $\pi_{\mathbf{v}}(z_1, z_2)$ for all $(z_1, z_2) \in \mathbb{Z}$ becomes 0 if $k_1^{(j)} + k_3^{(j)}$ is more than ℓ .

4.3 Path Search Algorithm for (ℓ, d) -Feistel

This section shows the path search algorithm for integral distinguishers against (ℓ, d) -Feistel. The algorithm is based on the propagation characteristic shown

Algorithm 1. Path search algorithm for integral distinguishers on (ℓ, d) -Feistel

```

1: procedure FeistelFuncEval( $\ell, d, k_1, k_2$ )
2:    $q \Leftarrow 0$ 
3:   for  $X = 0$  to  $k_1$  do
4:      $L \Leftarrow k_2 + \lceil X/d \rceil$ 
5:     if  $L \leq \ell$  then
6:        $q \Leftarrow q + 1$ 
7:        $\mathbf{k}^{(q)} \Leftarrow (L, k_1 - X)$ 
8:     end if
9:   end for
10:  return  $\mathbf{k}^{(1)}, \dots, \mathbf{k}^{(q)}$ 
11: end procedure

12: procedure IntegralPathSearch( $\ell, d, r = 0, k_1, k_2$ )
13:    $\mathbf{k}^{(1)}, \dots, \mathbf{k}^{(q)} \Leftarrow \text{FeistelFuncEval}(\ell, d, k_1, k_2)$ 
14:    $D \Leftarrow \max\{k_1^{(1)} + k_2^{(1)}, k_1^{(2)} + k_2^{(2)}, \dots, k_1^{(q)} + k_2^{(q)}\}$ 
15:   while  $1 < D$  do
16:      $r \Leftarrow r + 1$ 
17:     for  $i = 1$  to  $q$  do
18:        $\mathbf{k}^{(i,1)}, \dots, \mathbf{k}^{(i,p_i)} \Leftarrow \text{FeistelFuncEval}(\ell, d, k_1^{(i)}, k_2^{(i)})$ 
19:     end for
20:      $(\mathbf{k}^{(1)}, \mathbf{k}^{(2)}, \dots, \mathbf{k}^{(q')}) \Leftarrow \text{SizeReduce}(\mathbf{k}^{(1,1)}, \mathbf{k}^{(1,2)}, \dots, \mathbf{k}^{(q,p_q)})$ 
21:      $D \Leftarrow \max\{k_1^{(1)} + k_2^{(1)}, k_1^{(2)} + k_2^{(2)}, \dots, k_1^{(q')} + k_2^{(q')}\}$ 
22:      $q \Leftarrow q'$ 
23:   end while
24:   return  $r$ 
25: end procedure

```

in Sect. 4.2. Assume that k_1 bits of the left half of the input are active and the rest $(\ell - k_1)$ bits are constant. Moreover, assume that k_2 bits of the right half of the input are active and the rest $(\ell - k_2)$ bits are constant. Namely, we prepare $2^{k_1+k_2}$ chosen plaintexts. The input set has the division property $\mathcal{D}_{[k_1, k_2]}^{\ell, 2}$. Algorithm 1 shows the path search algorithm to create the integral distinguisher on (ℓ, d) -Feistel. Algorithm 1 does not limit the F -function to be a permutation. If the F -function is limited to be a permutation, L becomes $k_2 + \ell$ when $X = \ell$ holds (see the 4-th line in Algorithm 1). Algorithm 1 calls **SizeReduce**, which eliminates $\mathbf{k}^{(i,j)}$ if there exists (i', j') satisfying $\mathbb{S}_{\mathbf{k}^{(i,j)}}^{\ell, 2} \subseteq \mathbb{S}_{\mathbf{k}^{(i',j')}}^{\ell, 2}$.

Results. Table 2 shows the number of required chosen plaintexts to construct r -round integral distinguishers on $(32, 5)$ - and $(64, 7)$ -Feistel, where DES [31] is classified into $(32, 5)$ -Feistel with non-bijective function and Camellia [3] is classified into $(64, 7)$ -Feistel with bijective function. When we construct the integral distinguisher on (ℓ, d) -Feistel with 2^D chosen plaintexts, we use (k_1, k_2) satisfying

Table 2. The number of chosen plaintexts to construct r -round integral distinguishers on (32, 5)- and (64, 7)-Feistel. Our distinguishers are got by implementing Algorithm 1.

Target [Application]	F -function	$\log_2(\# \text{texts})$						Method	Reference
		$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$	$r = 9$		
(32, 5)-Feistel [DES]	non-bijection	26	51	62	-	-	-	our	Sect. 4.3
		26	-	-	-	-	-	degree	[8, 21]
(64, 7)-Feistel [Camellia]	bijection	50	98	124	-	-	-	our	Sect. 4.3
		50	-	-	-	-	-	degree	[8, 21]
		64	-	-	-	-	-	integral	[23]

Table 3. The number of chosen plaintexts to construct r -round integral distinguishers on the SIMON family, where the F -function is not bijective. Our distinguishers are got by implementing Algorithm 1.

Target [Application]	$\log_2(\# \text{texts})$								Method	Reference
	$r = 6$	$r = 7$	$r = 8$	$r = 9$	$r = 10$	$r = 11$	$r = 12$	$r = 13$		
(16, 2)-Feistel [SIMON 32]	17	25	29	31	-	-	-	-	our	Sect. 4.3
	-	-	-	-	-	-	-	-	degree	[8, 21]
(24, 2)-Feistel [SIMON 48]	17	29	39	44	46	47	-	-	our	Sect. 4.3
	17	-	-	-	-	-	-	-	degree	[8, 21]
(32, 2)-Feistel [SIMON 64]	17	33	49	57	61	63	-	-	our	Sect. 4.3
	17	-	-	-	-	-	-	-	degree	[8, 21]
(48, 2)-Feistel [SIMON 96]	17	33	57	77	87	92	94	95	our	Sect. 4.3
	17	33	-	-	-	-	-	-	degree	[8, 21]
(64, 2)-Feistel [SIMON 128]	17	33	65	97	113	121	125	127	our	Sect. 4.3
	17	33	-	-	-	-	-	-	degree	[8, 21]

$$(k_1, k_2) = \begin{cases} (D - \ell, \ell) & \text{for } \ell \leq D, \\ (0, D) & \text{for } D < \ell. \end{cases}$$

For the comparison with our integral distinguishers, we consider two previous methods, one is the propagation characteristic of the integral property and another is the estimation of the algebraic degree. We first consider the propagation characteristic of the integral property. If the F -function is a non-bijective function, the propagation characteristic does not construct sufficient distinguishers. Therefore, results introduced by the integral property are only shown when the F -function is bijective. We next consider the estimation of the algebraic degree. Unfortunately, since we do not know the improved bound against the Feistel Network, we use the trivial bound for the Feistel Network. Assume that the left half of the plaintext is constant. For any r -round (ℓ, d) -Feistel, it can be observed that the function, which associates the right half of the ciphertext with the right half of the plaintext, has degree at most d^{r-2} for $2 \leq r$. Therefore, we

can construct the r -round integral distinguishers with $2^{d^{r-2}+1}$ chosen plaintexts. Since the right half of the plaintext is at most ℓ bits, the distinguisher can be constructed with $2^{d^{r-2}+1} < 2^\ell$.

As a result, as far as we try, all distinguishers constructed by the division property are “better” than those by previous methods. We summarize integral distinguishers on other (ℓ, d) -Feistel in Appendix B. We already know a “better” integral distinguisher on Camellia in [36], but it is constructed by using the specific feature of Camellia. On the other hand, our method is generic distinguishing attacks against (ℓ, d) -Feistel. From the result of $(64, 7)$ -Feistel, it shows that even if the F -function of Camellia is chosen from any functions with degree 7, the modified Camellia has the 6-round integral distinguisher.

Integral Distinguishers on Simon Family. Although our attack is a generic attack, it can create new integral distinguishers on the SIMON family [5]. SIMON is a lightweight block ciphers proposed by the National Security Agency. Since SIMON has a non-bijective F -function and a bit-oriented structure, it is complicated task to construct the integral distinguisher. The division property theoretically shows that SIMON 32, 48, 64, 96, and 128 have at least 9-, 11-, 11-, 13-, and 13-round integral distinguishers, respectively. Table 3 shows the comparison between our distinguishers and previous ones by the degree estimation. On the other hand, Wang et al. showed that SIMON 32 has the 15-round integral distinguisher by experiments [33]. Therefore, there are 6-round differences between our theoretical result and Wang’s experimental result. Our distinguisher is valid against all $(32, 2)$ -Feistel and it does not exploit the feature of the round function. Namely, we expect that the 6-round difference is derived from the specification of the round function of SIMON 32.

5 Improved Integral Distinguishers on Substitute-Permutation Network

5.1 Substitute-Permutation Network

(ℓ, d, m) -SPN. The Substitute-Permutation Network (SPN) is another important structure for block ciphers. The SPN has a round function that consists of an S-Layer and a P-Layer, and a block cipher is designed by iterating the round function. We now define an (ℓ, d, m) -SPN, whose round function has m ℓ -bit S-boxes in the S-Layer and one (ℓm) -bit linear function in the P-Layer. Here, each S-box is any bijective function whose degree is at most d , and an (ℓm) -bit linear function is any bijective function whose degree is at most 1. Figure 6 shows the round function of the SPN. Nowadays, many block ciphers adopting (ℓ, d, m) -SPN have been proposed, e.g. AES [32], PRESENT [7], and Serpent [1] adopt $(8, 7, 16)$ -, $(4, 3, 16)$ -, and $(4, 3, 32)$ -SPN, respectively. Moreover, KECCAK- f [12], which is a permutation in the hash function KECCAK, can be regarded as $(5, 2, 320)$ -SPN.

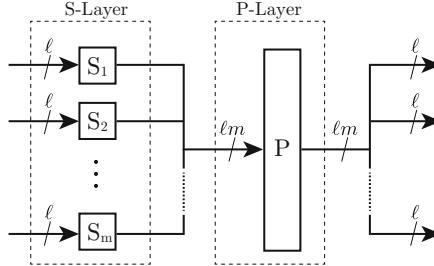


Fig. 6. (ℓ, d, m) -SPN

5.2 Propagation Characteristic for SPN

This section shows that the division property is useful to construct integral distinguishers on (ℓ, d, m) -SPN. We first prepare the set of the input of the S-Layer such that k_i bits of the input of the i -th S-box are active and the rest $(\ell - k_i)$ bits are constant. In this case, the input set has the division property $\mathcal{D}_{\mathbf{k}}^{\ell, m}$. We first evaluate the propagation characteristic against the S-Layer. Next, the P-Layer is applied but the input and output take a value of $\mathbb{F}_2^{\ell m}$. Therefore, we need to convert the division property $\mathcal{D}_{\mathbf{k}}^{\ell, m}$ into $\mathcal{D}_{\mathbf{k}'}^{\ell m}$, and then evaluate the propagation characteristic against the P-Layer. Since the S-Layer is applied again after the P-Layer, we convert the division property $\mathcal{D}_{\mathbf{k}'}^{\ell m}$ into $\mathcal{D}_{\mathbf{k}^{(1)}, \mathbf{k}^{(2)}, \dots, \mathbf{k}^{(q)}}^{\ell, m}$. After the second round, we evaluate the propagation characteristic of this collective division property.

- **S-Layer.** Assume that the input set of the S-Layer has the division property $\mathcal{D}_{\mathbf{k}}^{\ell, m}$. Since the S-Layer consists of m ℓ -bit S-boxes with degree d , the output set of the S-Layer has $\mathcal{D}_{\mathbf{k}'}^{\ell, m}$. Here, if $k_i < \ell$ holds, k'_i is calculated as $k'_i = \lceil k_i/d \rceil$. If $k_i = \ell$ holds, k'_i is calculated as $k'_i = \ell$.
- **Concatenation (Conversion form S-Layer to P-Layer).** The output of the S-Layer is expressed in a value of $(\mathbb{F}_2^\ell)^m$, but the input of the P-Layer is expressed in a value of $\mathbb{F}_2^{\ell m}$. Let \mathbb{X} be the output set of the S-Layer whose elements take a value of $(\mathbb{F}_2^\ell)^m$. Let \mathbb{Y} be the input set of the P-Layer whose elements take a value of $\mathbb{F}_2^{\ell m}$. The transformation is generally implemented by a simple bit concatenation, namely, $y = (x_1 \| x_2 \| \dots \| x_m)$ where (x_1, x_2, \dots, x_m) and y are values of \mathbb{X} and \mathbb{Y} , respectively. We now consider the conversion of the division property from $\mathcal{D}_{\mathbf{k}}^{\ell, m}$ to $\mathcal{D}_{\mathbf{k}'}^{\ell m}$. The parity of $\pi_v(y)$ for all $y \in \mathbb{Y}$ becomes unknown if and only if we choose v satisfying $w_v \geq \sum_{i=1}^m k_i$. Therefore, the input set of the P-Layer has the division property $\mathcal{D}_{\mathbf{k}'}^{\ell m}$, where $k' = \sum_{i=1}^m k_i$ holds.
- **P-Layer.** The P-Layer consists of an (ℓm) -bit linear function. Since the degree of the linear function is at most 1, there is no change in the division property.
- **Partition (Conversion form P-Layer to S-Layer).** The output of the P-Layer is expressed in a value of $\mathbb{F}_2^{\ell m}$, but the input of the S-Layer is expressed

in a value of $(\mathbb{F}_2^\ell)^m$. Let \mathbb{X} be the output set of the P-Layer whose elements take a value of $\mathbb{F}_2^{\ell m}$. Let \mathbb{Y} be the input set of the S-Layer whose elements take a value of $(\mathbb{F}_2^\ell)^m$. The transformation is generally implemented by a simple bit partition, namely, $(y_1 \| y_2 \| \cdots \| y_m) = x$ where x and (y_1, y_2, \dots, y_m) are values of \mathbb{X} and \mathbb{Y} , respectively. We now consider the conversion of the division property from $\mathcal{D}_k^{\ell m}$ to $\mathcal{D}_{k'}^{\ell, m}$. When the output set of the P-Layer has $\mathcal{D}_k^{\ell m}$, the sufficient condition that the parity of $\pi_u(x)$ for all $x \in \mathbb{X}$ becomes unknown is $k \leq w_u$. Therefore, the input set of the S-Layer has the collective division property $\mathcal{D}_{k'(1), k'(2), \dots, k'(q)}^{\ell, m}$, where q denotes the number of all possible vectors satisfying $k_1'^{(j)} + k_2'^{(j)} + \cdots + k_m'^{(j)} = k$ ($1 \leq j \leq q$). After the second round, we evaluate the propagation characteristic of the collective division property.

We can construct the integral distinguisher by evaluating the propagation characteristic of the collective division property. However, since the size of q extremely expands, it is infeasible to execute the straightforward implementation. Therefore, we show more efficient technique. Let \mathbb{X} be the input set of the S-Layer, and the elements take a value of $(\mathbb{F}_2^\ell)^m$. Assume that the input set has the division property $\mathcal{D}_{k^{(1)}, k^{(2)}, \dots, k^{(q)}}^{\ell, m}$ that is created by the partition of the division property $\mathcal{D}_k^{\ell m}$. If $k > (\ell - 1)m$ holds, at least $(m - \ell m + k)$ elements of $\mathbf{k}^{(j)}$ have to become ℓ . In this case, the rest elements have to become $\ell - 1$. Since the S-Layer derives $\lceil \frac{\ell-1}{d} \rceil$ and ℓ from $(\ell - 1)$ and ℓ , respectively, the output set has the division property $\mathcal{D}_{k'}^{\ell m}$, where k' is calculated as

$$k' = \begin{cases} \left\lceil \frac{\ell-1}{d} \right\rceil (\ell m - k) + \ell(m - \ell m + k) & \text{for } k > (\ell - 1)m, \\ \left\lceil \frac{k}{d} \right\rceil & \text{for } k \leq (\ell - 1)m. \end{cases}$$

Here, if $k \leq (\ell - 1)m$ holds, we simply regard the round function of (ℓ, d, m) -SPN as one (ℓm) -bit S-box with degree d .

5.3 Path Search Algorithm for (ℓ, d, m) -SPN

We now consider integral distinguishers on (ℓ, d, m) -SPN. We first prepare the set of chosen plaintexts such that k_i bits of the input of the i -th S-box are active and the rest $(\ell - k_i)$ bits are constant. Namely, we prepare $2^{\sum_{i=1}^m k_i}$ chosen plaintexts. The input set has the division property $\mathcal{D}_k^{\ell, m}$. Algorithm 2 shows the path search algorithm to construct the integral distinguisher.

Results. Table 4 shows the number of required chosen plaintexts to construct the r -round integral distinguisher on $(4, 3, 16)$ - and $(8, 7, 16)$ -SPN, where PRESENT [7] and AES [32] are classified into $(4, 3, 16)$ - and $(8, 7, 16)$ -SPN, respectively. When we construct the integral distinguisher on (ℓ, d, m) -SPN with 2^D chosen plaintexts, we use a vector \mathbf{k} satisfying

Algorithm 2. Path search algorithm for integral distinguishers on (ℓ, d, m) -SPN

```

1: procedure IntegralPathSearch( $\ell, d, m, r = 0, k_1, k_2, \dots, k_m$ )
2:   if  $k_i < \ell$  then  $k_i \leftarrow \lceil k_i/d \rceil$                                  $\triangleright$  1-st round S-Layer
3:   end if
4:    $k \leftarrow \sum_{i=1}^m k_i$                                                $\triangleright$  1-st round Concatenation and P-Layer
5:   while  $1 < k$  do
6:      $r \leftarrow r + 1$ 
7:     if  $k \leq (\ell - 1)m$  then  $k \leftarrow \lceil k/d \rceil$                                  $\triangleright (r + 1)\text{-th round}$ 
8:     else  $k \leftarrow \lceil \frac{\ell-1}{d} \rceil (\ell m - k) + \ell(m - \ell m + k)$        $\triangleright (r + 1)\text{-th round}$ 
9:     end if
10:    end while
11:    return  $r$ 
12: end procedure

```

Table 4. The number of chosen plaintexts to construct r -round integral distinguishers on (ℓ, d, m) -SPN. Our distinguishers are got by implementing Algorithm 2.

Target	$\log_2(\#\text{texts})$					Method	Reference
	$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$		
(4, 3, 16)-SPN [PRESENT]	12	28	52	60	-	our	Sect. 5.3
	28	52	60	63	-	degree	[9]
(8, 7, 16)-SPN [AES]	56	120	-	-	-	our	Sect. 5.3
	117	127	-	-	-	degree	[9]

Table 5. The number of chosen plaintexts to construct r -round integral distinguishers on KECCAK- f and Serpent. Our distinguishers are got by implementing Algorithm 2.

Target [Application]	$\log_2(\#\text{texts})$								Method	Reference
	$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$	$r = 9$	$r = 10$		
(4, 3, 32)-SPN [Serpent]	12	28	84	113	124	-	-	-	our	Sect. 5.3
	28	82	113	123	127	-	-	-	degree	[9]
Target [Application]	$\log_2(\#\text{texts})$								Method	Reference
	$r = 8$	$r = 9$	$r = 10$	$r = 11$	$r = 12$	$r = 13$	$r = 14$	$r = 15$		
(5, 2, 320)-SPN [KECCAK- f]	130	258	515	1025	1410	1538	1580	1595	our	Sect. 5.3
	257	513	1025	1409	1537	1579	1593	1598	degree	[9]

$$k_i = \begin{cases} \ell & \text{for } i\ell \leq D, \\ D - (i-1)\ell & \text{for } (i-1)\ell \leq D < i\ell, \\ 0 & \text{for } D < (i-1)\ell. \end{cases}$$

For the comparison with our integral distinguishers, we first consider the propagation characteristic of the integral property. However, it does not construct a sufficient distinguisher because the P-Layer is any linear function. Next,

we estimate the algebraic degree by using the method proposed by Boura et al. We show the method in Appendix A.

As a result, as far as we try, all distinguishers constructed by the division property are “better” than those by previous methods. We summarize integral distinguishers on other (ℓ, d, m) -SPN in Appendix C. We already know the 7-round integral distinguisher on PRESENT in [34] and the 4-round integral distinguisher on AES in [23]. However, they are constructed by using the specific feature of each block cipher. On the other hand, our method is generic distinguishing attacks against (ℓ, d, m) -SPN. From the result of $(4, 3, 16)$ -SPN, it shows that even if the P-Layer of PRESENT is chosen from any bijective linear functions, the modified PRESENT has the 6-round integral distinguisher. Similarly, from the result of $(8, 7, 16)$ -SPN, it shows that even if the P-Layer of AES is chosen from any bijective linear function, the modified AES still has the 4-round integral distinguisher.

Integral Distinguishers on Serpent and Keccak- f Although our attack is a generic attack, it can create new integral distinguishers on Serpent and KECCAK- f . Serpent is one of AES finalists and is classified into $(4, 3, 32)$ -SPN. The existing integral distinguisher is shown in [37], and it shows that Serpent has 3.5-round integral distinguisher. On the other hand, we show that all $(4, 3, 32)$ -SPNs have at least 7-round integral distinguishers with 2^{124} chosen plaintexts. Table 5 shows the comparison between our distinguishers and previous ones by the degree estimation.

KECCAK is chosen as SHA-3, and the core function KECCAK- f is classified into $(5, 2, 320)$ -SPN. Boura et al. estimated the algebraic degree of KECCAK- f in [9]. We search for the integral distinguisher by using Algorithm 2. As a result, our distinguishers can reduce the number of chosen plaintexts compared with previous ones. Table 5 shows the comparison between our distinguishers and previous ones.

6 Toward Dedicated Attack

We introduced the division property in Sect. 3, and proposed distinguishing attacks against the Feistel Network and the SPN in Sect. 4 and Sect. 5, respectively. In this section, we show that the division property is also useful to construct the dedicated attack against specific ciphers. As an example, we show integral distinguishers on AES-like ciphers.

6.1 AES-Like Cipher

(ℓ, d, m) -AES. AES is a 128-bit block cipher, and an intermediate text of AES is expressed in a 4×4 matrix whose elements are 8 bits. The round function of AES consists of SubBytes, ShiftRows, MixColumns, and AddRoundKey, where each function is defined as follows:

Algorithm 3. Evaluating algorithm against the round function of (ℓ, d, m) -AES

```

1: procedure AesFuncEval( $\ell, d, m, K$ )
2:   for  $r = 1$  to  $m$  do
3:     for  $c = 1$  to  $m$  do
4:       if  $k_{r,c} < \ell$  then  $k_{r,c} \leftarrow \lceil k_{r,c}/d \rceil$                                  $\triangleright$  SubBytes
5:       end if
6:     end for
7:   end for
8:    $K \leftarrow \text{ShiftRows}(K)$                                                $\triangleright$  ShiftRows
9:    $k'_c \leftarrow \sum_{r=1}^m k'_{r,c}$  for all  $c$                                           $\triangleright$  MixColumns
10:   $k' \leftarrow \text{sort}(k')$ 
11:  return  $k'$ 
12: end procedure

```

- SubBytes (SB) : It substitutes each byte in the matrix into another byte by an S-box.
- ShiftRows (SR) : Each byte of the i -th row is rotated $i - 1$ bytes to the left.
- MixColumns (MC) : It diffuses bytes within each column by a linear function.
- AddRoundKey (AK) : A round key is XORed with the intermediate text.

We define an (ℓ, d, m) -AES, where ℓ , d , and m denote the bit length of an S-box, the algebraic degree of an S-box, and the size of the matrix, respectively. This intermediate text is expressed in an $m \times m$ matrix whose elements are ℓ bits. Let $\mathbf{X} \in (\mathbb{F}_2^\ell)^{m \times m}$ be an input of the round function, which is arranged as

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,m} \end{bmatrix}.$$

Let $\mathbf{Y} \in (\mathbb{F}_2^\ell)^{m \times m}$ be an output of the round function, which is calculated as $\mathbf{Y} = (\text{AK} \circ \text{MC} \circ \text{SR} \circ \text{SB})(\mathbf{X})$. Each function is the same as that of AES except for the scale. For instance, AES [32] and LED [18] adopt $(8, 7, 4)$ -AES and $(4, 3, 4)$ -AES, respectively. Moreover, P_{256} of PHOTON [17] adopts $(4, 3, 8)$ -AES².

6.2 Path Search Algorithm for (ℓ, d, m) -AES

Section 5 shows how to construct integral distinguishers on (ℓ, d, m) -SPN, but practical block ciphers have a specific P-Layer. For instance, the P-Layer in AES consists of ShiftRows and MixColumns, and it is not any linear function. Taking into account the structure of the P-Layer, we can construct more effective algorithm. In this section, as an example, we show a path search algorithm to construct integral distinguishers on (ℓ, d, m) -AES. Algorithm 3 evaluates the propagation characteristic of the division property against the round function

² Since PHOTON is a hash function, it uses AddConstant instead of AddRoundKey.

Algorithm 4. Path search algorithm for integral distinguishers on (ℓ, d, m) -AES

```

1: procedure IntegralPathSearch( $\ell, d, m, r = 0, \mathbf{K} \in \{0, 1, \dots, \ell\}^{m \times m}$ )
2:    $\mathbf{k}^{(1)} \leftarrow \text{AesFuncEval}(\ell, d, m, \mathbf{K})$                                  $\triangleright$  1-st round
3:    $D \leftarrow \sum_{c=1}^m k_c^{(1)}$ 
4:    $q \leftarrow 1$ 
5:   while  $1 < D$  do
6:      $r \leftarrow r + 1$ 
7:     for  $i = 1$  to  $q$  do
8:        $\mathbf{K}^{(i,1)}, \dots, \mathbf{K}^{(i,s)} \leftarrow \text{Partition}(\mathbf{k}^{(i)})$ 
9:       for  $j = 1$  to  $s$  do
10:         $\bar{\mathbf{k}}^{(1)}, \dots, \bar{\mathbf{k}}^{(t)} \leftarrow \text{AesFuncEval}(\ell, d, m, \mathbf{K}^{(i,j)})$        $\triangleright (r+1)\text{-th round}$ 
11:        if  $(i, j) = (1, 1)$  then
12:           $\mathbf{k}'^{(1)}, \dots, \mathbf{k}'^{(q')}$   $\leftarrow \text{SizeReduce}(\bar{\mathbf{k}}^{(1)}, \dots, \bar{\mathbf{k}}^{(t)})$ 
13:        else
14:           $\mathbf{k}'^{(1)}, \dots, \mathbf{k}'^{(q'')} \leftarrow \text{SizeReduce}(\mathbf{k}'^{(1)}, \dots, \mathbf{k}'^{(q')}, \bar{\mathbf{k}}^{(1)}, \dots, \bar{\mathbf{k}}^{(t)})$ 
15:           $q' \leftarrow q''$ 
16:        end if
17:      end for
18:    end for
19:     $\mathbf{k}^{(i)} \leftarrow \mathbf{k}'^{(i)}$  for all  $1 \leq i \leq q'$ 
20:     $q \leftarrow q'$ 
21:     $D \leftarrow \min\{\sum_{c=1}^m k_c^{(1)}, \sum_{c=1}^m k_c^{(2)}, \dots, \sum_{c=1}^m k_c^{(q)}\}$ 
22:  end while
23:  return  $r$ 
24: end procedure

```

of AES-like ciphers, and it calls `ShiftRows` and `sort`. `ShiftRows` performs a similar transformation to SR. `sort` is the sorting algorithm, which is useful for feasible implementation. Algorithm 4 shows the path search algorithm, and it calls `Partition`, `AesFuncEval`, and `SizeReduce`. `Partition`($\mathbf{k}^{(i)}$) calculates all possible $\mathbf{K}^{(i,j)}$ satisfying

$$\left(\sum_{r=1}^m k_{r,1}^{(i,j)}, \sum_{r=1}^m k_{r,2}^{(i,j)}, \dots, \sum_{r=1}^m k_{r,m}^{(i,j)} \right) = (k_1^{(i)}, k_2^{(i)}, \dots, k_m^{(i)}),$$

where $0 \leq k_{r,c}^{(i,j)} \leq \ell$ holds. `SizeReduce` eliminates $\mathbf{k}^{(i,j)}$ if there exists (i', j') satisfying $\mathbb{S}_{\mathbf{k}^{(i,j)}}^{\ell m, m} \subseteq \mathbb{S}_{\mathbf{k}^{(i',j')}}^{\ell m, m}$.

Notice that the size of q in the division property extremely expands when the partition of the division property is executed (see the 8-th line in Algorithm 4). Namely, our algorithm takes large execution time and large memory capacity if we straightforwardly implement our algorithm. Therefore, we use an effective method, which uses the feature of (ℓ, d, m) -AES, for the feasible implementation. Notice that each column of (ℓ, d, m) -AES is equivalent each other. Assuming that the input set has $\mathcal{D}_{\mathbf{k}, \mathbf{k}'}^{\ell m, m}$ that \mathbf{k}' is a permutation of elements of \mathbf{k} , the division property of the next round calculated from \mathbf{k} is exactly the same as that from \mathbf{k}' because columns of (ℓ, d, m) -AES are equivalent each other. Namely, it is enough

Table 6. The number of chosen plaintexts to construct r -round integral distinguishers on $(4, 3, m)$ -AES. Our distinguishers are got by implementing Algorithm 2 and Algorithm 4.

Target [Application]	$\log_2(\# \text{texts})$						Method	Reference
	$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$		
(4, 3, 4)-AES [LED]	4	12	32	52	-	-	our (AES)	Sect. 6.2
	12	28	52	60	-	-	our (SPN)	Sect. 5.3
	28	52	60	63	-	-	degree	[9]
	4	16	-	-	-	-	integral	[13, 23]
(4, 3, 5)-AES [P_{100} in PHOTON]	4	12	20	72	97	-	our (AES)	Sect. 6.2
	12	28	76	92	-	-	our (SPN)	Sect. 5.3
	28	76	92	98	-	-	degree	[9]
	4	20	-	-	-	-	integral	[13, 23]
(4, 3, 6)-AES [P_{144} in PHOTON]	4	12	24	84	132	-	our (AES)	Sect. 6.2
	12	28	84	124	140	-	our (SPN)	Sect. 5.3
	28	82	124	138	142	-	degree	[9]
	4	24	-	-	-	-	integral	[13, 23]
(4, 3, 7)-AES [P_{196} in PHOTON]	4	12	24	84	164	192	our (AES)	Sect. 6.2
	12	28	84	160	184	192	our (SPN)	Sect. 5.3
	28	82	158	184	192	195	degree	[9]
	4	28	-	-	-	-	integral	[13, 23]
(4, 3, 8)-AES [P_{256} in PHOTON]	4	12	28	92	204	249	our (AES)	Sect. 6.2
	12	28	84	200	237	252	our (SPN)	Sect. 5.3
	28	82	198	237	250	254	degree	[9]
	4	32	-	-	-	-	integral	[13, 23]

to save either, and we implement it by a sorting algorithm (see the 10-th line in Algorithm 3). This technique enables us to execute our path search algorithm feasibly in many parameters.

Results. Table 6 shows the number of required chosen plaintexts to construct r -round integral distinguishers on $(4, 3, m)$ -AES. When we construct the integral distinguisher on (ℓ, d, m) -AES with 2^D chosen plaintexts, we carefully choose the input matrix \mathbf{K} .

For the comparison with our improved integral distinguishers, we also show integral distinguishers by using the propagation characteristic of the integral property. We also estimate the algebraic degree by the method proposed Boura et al. (see Appendix A). Moreover, since $(4, 3, m)$ -AES are classified into $(4, 3, m^2)$ -SPN, we construct integral distinguishers by Algorithm 2.

As a result, as far as we try, all distinguishers constructed by the division property are at least better than those by previous methods. Especially, the advantage of our method is large when we construct the integral distinguisher with the small number of texts. For instance, our method shows that $(4, 3, 8)$ -AES, which is adopted by P_{256} in PHOTON, has the 6-round distinguisher with

2^{92} chosen plaintexts. If we regard $(4, 3, 8)$ -AES as $(4, 3, 64)$ -SPN, 2^{200} chosen plaintexts are required to construct the distinguisher.

7 Conclusions

In this paper, we proposed the fundamental technique to improve integral distinguishers, and showed structural cryptanalyses against the Feistel Network and the SPN. Our new technique uses the division property, which is the generalization of the integral property. It can effectively construct integral distinguishers even if block ciphers have non-bijective functions, bit-oriented structures, and low-degree functions. For the Feistel Network, when the algebraic degree of the F -function is smaller than the bit length of the F -function, our method can attack more rounds than previous generic attacks. Moreover, we theoretically showed that SIMON 48, 64, 96, and 128 have 11-, 11-, 13-, and 13-round integral distinguishers, respectively. For the SPN, our method extremely reduces the required number of chosen plaintexts compared with previous methods. Moreover, we improved integral distinguishers on KECCAK- f and Serpent. The division property is useful to construct integral distinguishers against specific ciphers. As one example, we showed a path search algorithm to construct integral distinguishers on the AES-like cipher, which is the sub class of the SPN. From this fact, we expect that the division property can construct many improved integral distinguishers against specific ciphers by constructing the dedicated path search algorithm.

A Estimation of Algebraic Degree for (ℓ, d, m) -SPN

If the degree of r iterated round functions is at most D , we can construct the r -round integral distinguisher with 2^{D+1} chosen plaintexts. In a classical method, if the degree of the round function is at most d , the degree of r iterated round functions is bounded by d^r . In 2011, Boura et al. showed tighter bound as follows.

Theorem 1 ([9]). *Let S be a function from \mathbb{F}_2^n into \mathbb{F}_2^n corresponding to the concatenation of m smaller S -boxes, defined over $\mathbb{F}_2^{n_0}$. Let δ_k be the maximal degree of the product of any k bits of anyone of these S -boxes. Then, for any function G from \mathbb{F}_2^n into \mathbb{F}_2 , we have*

$$\deg(G \circ S) \leq n - \frac{n - \deg(G)}{\gamma},$$

where

$$\gamma = \max_{1 \leq i \leq n_0 - 1} \frac{n_0 - i}{n_0 - \delta_i}.$$

By using this bound, we can estimate the degree of (ℓ, d, m) -SPN. For instance, we show the degree of $(4, 3, 64)$ -SPN as follows.

Number of rounds	1	2	3	4	5	6	7	8	9
Bound on degree	3	9	27	81	197	236	249	253	255

Therefore, we can construct the 8-round integral distinguisher on $(4, 3, 64)$ -SPN with 2^{254} chosen plaintexts.

B Integral Distinguishers on (ℓ, d) -Feistel

Table 7 shows integral distinguishers on (ℓ, d) -Feistel, where (ℓ, d) -Feistel is defined in Sect. 4.1. If we construct the dedicated path search algorithm for

Table 7. The number of required chosen plaintexts to construct r -round integral distinguishers on (ℓ, d) -Feistel. We get these values by implementing Algorithm 1.

the specific cipher, we expect that the algorithm can create better integral distinguishers.

C Integral Distinguishers on (ℓ, d, m) -SPN

Table 8 shows integral distinguishers on (ℓ, d, m) -SPN, where (ℓ, d, m) -SPN is defined in Sect. 5.1. If we construct the dedicated path search algorithm for the specific cipher, we expect that the algorithm can create better integral distinguishers.

Table 8. The number of required chosen plaintexts to construct r -round integral distinguishers on (ℓ, d, m) -SPN. We get these values by implementing Algorithm 2.

Target	Size (bits)	$\log_2(\# \text{texts})$							Examples
		$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$	$r = 9$	$r = 10$	
(4, 3, 16)	64	28	52	60	-	-	-	-	PRESENT [7], LED [18]
(4, 3, 24)	96	28	76	89	-	-	-	-	
(4, 3, 32)	128	28	84	113	124	-	-	-	Serpent [1], NOEKEON [14]
(4, 3, 40)	160	28	84	136	152	-	-	-	
(4, 3, 48)	192	28	84	156	180	188	-	-	
(4, 3, 56)	224	28	84	177	209	220	-	-	
(4, 3, 64)	256	28	84	200	237	252	-	-	Prøst-128 [20], Minalpher- P [29]
(4, 3, 128)	512	28	84	244	424	484	504	509	Prøst-256 [20]
Target	Size (bits)	$\log_2(\# \text{texts})$							Examples
		$r = 5$	$r = 6$	$r = 7$	$r = 8$	$r = 9$	$r = 10$	$r = 11$	
(5, 2, 40)	200	18	35	65	130	178	195	-	PRIMATE-80 [2]
(5, 2, 56)	280	18	35	65	130	230	265	275	PRIMATE-120 [2]
(5, 2, 64)	320	18	35	65	130	258	300	315	ASCON Permutation [16]
Target	Size (bits)	$\log_2(\# \text{texts})$							Examples
		$r = 9$	$r = 10$	$r = 11$	$r = 12$	$r = 13$	$r = 14$	$r = 15$	
(5, 2, 160)	800	258	515	705	770	790	798	-	KECCAK-f[800] [12]
(5, 2, 256)	1280	258	515	1025	1195	1253	1271	1278	
(5, 2, 320)	1600	258	515	1025	1410	1538	1580	1595	KECCAK-f[1600] [12]
Target	Size (bits)	$\log_2(\# \text{texts})$							Examples
		$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$	$r = 9$	
(5, 4, 40)	200	20	65	170	195	-	-	-	
(5, 4, 56)	280	20	65	230	270	-	-	-	
(5, 4, 64)	320	20	65	260	305	-	-	-	
(5, 4, 160)	800	20	65	260	665	770	795	-	
(5, 4, 256)	1280	20	65	260	1025	1220	1265	-	ICEPOLE Permutation [27]
(5, 4, 320)	1600	20	65	260	1025	1460	1565	1595	
Target	Size (bits)	$\log_2(\# \text{texts})$							Examples
		$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$	$r = 9$	
(8, 7, 16)	128	56	120	-	-	-	-	-	AES [32]
(8, 7, 24)	192	56	176	-	-	-	-	-	Rijndael-192 [15]
(8, 7, 32)	256	56	232	-	-	-	-	-	Rijndael-256 [15]
(8, 7, 64)	512	56	344	488	-	-	-	-	WHIRLPOOL primitive [4]

References

1. Anderson, R., Biham, E., Knudsen, L.: Serpent: A proposal for the Advanced Encryption Standard. NIST AES Proposal (1998)
2. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., Yasuda, K.: PRIMATEs v1.02 (2014), submission to CAESAR competition
3. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-Bit block cipher suitable for multiple platforms - design and analysis. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 39–56. Springer, Heidelberg (2001)
4. Barreto, P.S.L.M., Rijmen, V.: The Whirlpool hashing function (2003), submitted to the NESSIE project. <http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html>
5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptology ePrint Archive 2013, 404 (2013). <http://eprint.iacr.org/2013/404>
6. Biryukov, A., Shamir, A.: Structural cryptanalysis of SASAS. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 395–405. Springer, Heidelberg (2001)
7. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
8. Boura, C., Canteaut, A.: On the influence of the algebraic degree of F^{-1} on the algebraic degree of $G \circ F$. IEEE Transactions on Information Theory **59**(1), 691–702 (2013)
9. Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of KECCAK and Luffa. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011)
10. Cannière, C.D., Sato, H., Watanabe, D.: Hash function Luffa - a SHA-3 candidate (2008). http://hitachi.com/rd/yr1/crypto/luffa/round1archive/Luffa_Specification.pdf
11. Canteaut, A., Videau, M.: Degree of composition of highly nonlinear functions and applications to higher order differential cryptanalysis. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 518–533. Springer, Heidelberg (2002)
12. Daemen, J., Bertoni, G., Peeters, M., Assche, G.V.: The Keccak reference version 3.0 (2011)
13. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
14. Daemen, J., Peeters, M., Assche, G.V., Rijmen, V.: The Noekeon block cipher. (2000), submitted to the NESSIE project. <http://gro.noekeon.org/>
15. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002)
16. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1 (2014), submission to CAESAR competition
17. 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)
18. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)

19. Isobe, T., Shibutani, K.: Generic key recovery attack on Feistel scheme. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 464–485. Springer, Heidelberg (2013)
20. Kavun, E.B., Lauridsen, M.M., Leander, G., Rechberger, C., Schwabe, P., Yalçın, T.: Prøst v1.1 (2014), submission to CAESAR competition
21. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
22. Knudsen, L.R.: The security of Feistel ciphers with six rounds or less. *J. Cryptology* **15**(3), 207–222 (2002)
23. Knudsen, L.R., Wagner, D.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002)
24. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Communications and Cryptography. The Springer International Series in Engineering and Computer Science, vol. 276, pp. 227–233 (1994)
25. Li, Y., Wu, W., Zhang, L.: Improved integral attacks on reduced-round CLEFIA block cipher. In: Jung, S., Yung, M. (eds.) WISA 2011. LNCS, vol. 7115, pp. 28–39. Springer, Heidelberg (2012)
26. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.* **17**(2), 373–386 (1988)
27. Morawiecki, P., Gaj, K., Homsirikamol, E., Matusiewicz, K., Pieprzyk, J., Rogawski, M., Srebrny, M., Wójcik, M.: ICEPOLE v1 (2014), submission to CAESAR competition
28. Patarin, J.: Security of random Feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 106–122. Springer, Heidelberg (2004)
29. Sasaki, Y., Todo, Y., Aoki, K., Naito, Y., Sugawara, T., Murakami, Y., Matsui, M., Hirose, S.: Minalpher v1 (2014), submission to CAESAR competition
30. Shibayama, N., Kaneko, T.: A peculiar higher order differential of CLEFIA. In: ISITA, pp. 526–530. IEEE (2012)
31. National Institute of Standards and Technology: Data Encryption Standard (DES). Federal Information Processing Standards Publication 46 (1977)
32. National Institute of Standards and Technology: Specification for the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standards Publication 197 (2001)
33. Wang, Q., Liu, Z., Varici, K., Sasaki, Y., Rijmen, V., Todo, Y.: Cryptanalysis of reduced-round SIMON32 and SIMON48. In: Daemen, J., Rijmen, V. (eds.) INDOCRYPT. LNCS, vol. 8885, pp. 143–160. Springer, Heidelberg (2014)
34. Wu, S., Wang, M.: Integral attacks on reduced-round PRESENT. In: Qing, S., Zhou, J., Liu, D. (eds.) ICICS 2013. LNCS, vol. 8233, pp. 331–345. Springer, Heidelberg (2013)
35. Wu, W., Zhang, L.: LBlock: A lightweight block cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)
36. Yeom, Y., Park, S., Kim, I.: On the security of CAMELLIA against the Square attack. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 89–99. Springer, Heidelberg (2002)
37. Z’aba, M.R., Raddum, H., Henricksen, M., Dawson, E.: Bit-pattern based integral attack. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 363–381. Springer, Heidelberg (2008)

Cryptanalysis of SP Networks with Partial Non-Linear Layers

Achiya Bar-On¹, Itai Dinur^{2(✉)}, Orr Dunkelman^{3,5}, Virginie Lallemand⁴,
Nathan Keller^{1,5}, and Boaz Tsaban¹

¹ Department of Mathematics, Bar-Ilan University, Ramat Gan, Israel

² Département d’Informatique, École Normale Supérieure, Paris, France

Itai.Dinur@ens.fr

³ Computer Science Department, University of Haifa, Haifa, Israel

⁴ Inria, Paris, France

⁵ Computer Science Department, The Weizmann Institute, Rehovot, Israel

Abstract. Design of SP networks in which the non-linear layer is applied to only a part of the state in each round was suggested by Gérard et al. at CHES 2013. Besides performance advantage on certain platforms, such a design allows for more efficient masking techniques that can mitigate side-channel attacks with a small performance overhead.

In this paper we present generic techniques for differential and linear cryptanalysis of SP networks with partial non-linear layers, including an automated characteristic search tool and dedicated key-recovery algorithms. Our techniques can be used both for cryptanalysis of such schemes and for proving their security with respect to basic differential and linear cryptanalysis, succeeding where previous automated analysis tools seem to fail.

We first apply our techniques to the block cipher Zorro (designed by Gérard et al. following their methodology), obtaining practical attacks on the cipher which were fully simulated on a single desktop PC in a few days. Then, we propose a mild change to Zorro, and formally prove its security against basic differential and linear cryptanalysis. We conclude that there is no inherent flaw in the design strategy of Gérard et al., and it can be used in future designs, where our tools should prove useful.

Keywords: Block cipher · Lightweight · Zorro · Differential cryptanalysis · Linear cryptanalysis

Achiya Bar-On—The research of the first author was partially supported by the Israeli Ministry of Science, Technology and Space, and by the Check Point Institute for Information Security.

Orr Dunkelman—The third author was supported in part by the German-Israeli Foundation for Scientific Research and Development through grant No. 2282-2222.6/2011.

Virginie Lallemand—The fourth author was partially supported by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011.

Nathan Keller—The fifth author was supported by the Alon Fellowship.

1 Introduction

Most block ciphers are either SP networks that apply linear and non-linear layers to the entire state in every encryption round, or (generalized) Feistel structures that apply partial linear and non-linear layers in every round. In the CHES 2013 paper [10], Gérard et al. suggested a compromise between the two common block cipher designs – an SP network in which the non-linear layer is applied to only a part of the state in every round. Such partial non-linear SP networks (which we call *PSP networks*) contain a wide range of possible concrete schemes that were not considered so far, some of which have performance advantage on certain platforms. More importantly, PSP networks allow for more efficient masking techniques, capable of thwarting side-channel attacks with a small performance overhead.

As a concrete instantiation of their methodology, Gérard et al. designed Zorro, a 128-bit lightweight block cipher. Zorro has an unconventional structure, as it applies a sequence of 24 AES-like rounds, with a partial S-box layer in each round, containing only 4 out of the possible 16 S-boxes. Since previous tools that were developed in order to formally prove the security of block ciphers against standard differential and linear cryptanalysis (such as the wide-trail strategy used for AES) do not apply to PSP networks such as Zorro, the authors replaced the formal proof for Zorro by a heuristic argument. Unfortunately, the heuristic argument turned out to be insufficient, as Wang et al. [16] found iterative differential and linear characteristics that were missed by the heuristic and used them to break full Zorro with complexity of 2^{112} .

In this paper, we propose efficient algorithms for differential and linear cryptanalysis of PSP networks. These algorithms allow us to fully evaluate the security of such constructions against standard differential and linear cryptanalysis. In some cases, we can compute tight upper bounds on the probability of differential and linear characteristics, thus offering formal proofs which are expected from any proposal of a modern block cipher.

Our most useful tool is a generic differential/linear characteristic search algorithm, allowing us to search for the best differential/linear characteristics for many rounds with a practical time complexity. A complementary tool is an efficient key recovery technique for differential and linear attacks, making use of the partial S-box layers to analyze more rounds at the end of the cipher with no increase in the attack's complexity.

1.1 Our New Automated Characteristic Search Tool

The starting point of our characteristic search algorithm is the algorithm of Biryukov and Nikolic [3] (along with several related algorithms, starting from Matsui's classical algorithm [12] and more recent ones [4, 13]), which is based on a compact representation of differential characteristics, that we call a *pattern*. At its most basic form, a pattern describes for each byte (or nibble) of the

cipher's state, whether it is active (namely, it has a non-zero input difference) or inactive.¹

Patterns allow the algorithm to group together and simultaneously analyze many characteristics for a given number of the cipher's rounds. The algorithm outputs only patterns that contain the smallest number of active S-boxes, and thus correspond to high probability characteristics. However, depending on the analyzed cipher, not all possible patterns are valid, as there are patterns not followed by any actual characteristic.² Thus, in order to provide meaningful results, a characteristic search algorithm has to ensure that it only outputs valid patterns.

Previous search algorithms [3, 4, 12, 13] indeed made sure that their output patterns were valid. This was done using local consistency checks, separately ensuring that for each of the r rounds of a pattern, there exist characteristics that satisfy the transitions of the round³ (i.e., conform to the 1-round pattern). For standard block ciphers, ensuring that an r -round pattern is *locally valid* (in the sense described above) also implies that it is *globally valid*, namely, there exists an actual r -round characteristic that simultaneously satisfies all the transitions of the r rounds.

Unlike standard block ciphers, for PSP networks there exist many locally valid patterns which are not globally valid over several rounds. In order to demonstrate this, consider a 4-round AES-like cipher with 4 S-boxes in each round (such as 4-round Zorro). The cipher contains a total of $4 \cdot 4 = 16$ S-boxes, and a larger number of $12 \cdot 4 = 48$ state bytes that do not go through an S-box in these rounds. It is easy to see that the cipher has a large number of locally valid patterns in which all the 16 S-boxes are inactive, as in each round, there are many valid active/inactive possibilities for the 12 bytes that do not go through an S-box. Consequently, when applying previous algorithms (such as [3]) to this cipher, we obtain many patterns in which all the 16 S-boxes are inactive, containing a huge number of possible 4-round characteristics with probability 1. However, as we show next, it is likely that none of these characteristics is globally valid, rendering previous algorithms ineffective for this (seemingly simple) PSP network.

At a high level, the reason that it is likely that there exists no characteristic in which all the 16 S-boxes are inactive, is that each inactive S-box forces the input difference to 0, imposing a constraint on the characteristic. Thus, for the 4-round cipher, we have 16 such constraints, whereas the number of available degrees of freedom to choose the input difference at the first round is also 16. Consequently, we have the same number of constraints and degrees of freedom, and it is probable that the constraints cannot be simultaneously satisfied (which is indeed the case for 4-round Zorro, as shown in [10]).

¹ For example, the 16 bytes of the 128-bit AES state can be described by a pattern of only 16 bits.

² For example, if the input to an AES round contains 1 active byte, then its output contains exactly 4 active bytes, and all other patterns are automatically invalid.

³ The algorithm of [4] is a bit different, as a characteristic is broken down into groups of 3 consecutive rounds.

In order to take into account global constraints, we group characteristics according to patterns similarly to previous algorithms. However, unlike previous algorithms, our patterns do not contain information about the full state, but only about the activity/inactivity of the bytes that go through S-boxes. Then, we observe that all the constraints imposed on a characteristic that follows such a pattern can be described by a set of linear equations. This observation allows us to group together and efficiently analyze, many characteristics which reside in a subspace, defined according to subtle linear constraints imposed by the cipher's linear layer.

Previous related automated search tools of [3, 4, 12, 13] mostly employed method of dynamic programming and mixed integer-linear programming. On the other hand, our characteristic search algorithm, as well as our key recovery algorithms, is mostly based on linearization techniques, which combine in a novel way methods from simple linear algebra and combinatorics, and may be of independent interest. These techniques exploit the small number of S-boxes in the non-linear layers of the cipher in order to “linearize” sequences of rounds, thus making it possible to analyze many rounds efficiently. We stress that while we focus in this paper on PSP networks, our algorithms can potentially offer new insights on the security of other designs that apply a non-linear function to only a part of the state in each round, such as (generalized) Feistel constructions and stream ciphers.

1.2 Main Application of the New Tool: Studying the Security of the PSP Network Design Methodology

As a first demonstration of our techniques, we apply them to the block cipher Zorro, improving the complexity of the previously best attack from 2^{112} to a practical 2^{45} . Our attack was fully simulated several times on a standard desktop PC over several days. This is a rare case in which an attack on a modern block cipher is fully simulated in practice.

More significantly, we address the general question of whether the attacks on Zorro indicate a structural flaw in its design methodology, or are merely a result of an unlucky combination of components. Our conclusion is that indeed the methodology of building PSP networks based on AES in a straightforward way is flawed, and should not be reused. The structural weakness in this methodology is due to a subtle inter-relation between the ShiftRows and MixColumns operations of AES, that may need to be taken into consideration in future designs, especially in light of the common practice of using *part* of the AES components as building blocks.

Finally, we address an even more general question of whether the basic PSP network design methodology is flawed, or it can potentially be reused in future designs. This question is investigated by analyzing a PSP network that slightly deviates from the AES design strategy, having a lightly modified ShiftRows mapping. We analyze this scheme using our characteristic search tool and formally prove its resistance to standard differential and linear cryptanalysis (as expected from modern block ciphers). Thus, as the most important application

of our tools, we answer the main question posed by this paper, concluding that PSP networks are not inherently flawed, and can be reused (with caution) to build secure block ciphers.

1.3 Organization of the Paper

We start by presenting our generic characteristic search algorithms for PSP networks in Section 2. Our generic key recovery algorithms for differential and linear attacks are given in Sections 3 and 4, respectively. In Section 5, we use our algorithms to attack Zorro. Finally, we study the problem of designing secure PSP networks in Section 6 and conclude in Section 7.

2 Generic High-Probability Characteristic Search Algorithm for PSP Networks

In this section we present a novel and efficient high-probability characteristic search algorithm for SP networks with partial non-linear layers. The search algorithm is only presented for differential characteristics, but we note that the algorithm for linear characteristics is very similar. As the algorithm is somewhat involved, we first describe it at a high-level and then present it in detail. Finally, we describe an optimization which is very useful in practice.

For ease of exposition, we describe the algorithm on the example of an AES-like cipher, in which a 128-bit state is represented as a 4-by-4 matrix of bytes, and the S-boxes (which are the only non-linear operation) act on some of the bytes in each round. The number of S-boxes in each round is denoted by t (e.g., $t = 16$ for AES and $t = 4$ for Zorro). Hence, we shall concentrate on a PSP that contains the following parts:

- S-box layer — The S-box layer is applied to t out of the 16 state bytes. The S-boxes are all invertible.
- Linear layer — The linear layer L is applied to the state. We do not assume anything in particular concerning the structure of L (as long as it is invertible).
- Key addition layer — XORing the subkey into the state.

As common in AES-like ciphers, we shall assume that there is one key addition layer before the first round (it does not affect our results whatsoever), and one can have in the last round a different linear layer (our described attacks and algorithms are trivially extended to cases where each round has its own linear layer).

Inspired by [3], we define the *pattern* of a differential (or a linear) characteristic to be a description of the activity for each of its spanned S-boxes (see Figure 1). Namely, a pattern is a function that specifies for each S-box spanned by the characteristic whether it is active (i.e., has a non-zero input difference) or not. We note that while [3] defines a pattern over the full bytes (S-boxes) of a state, we define it only over the bytes that are covered by S-boxes.

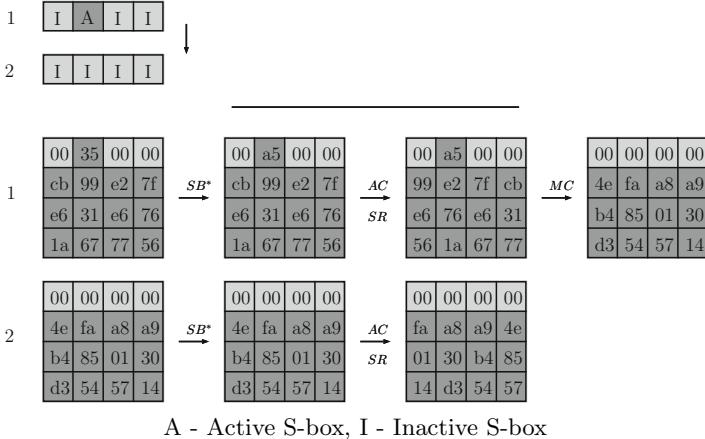


Fig. 1. A 2-Round Pattern and a Differential Characteristic that Follows it

2.1 An Overview of the Algorithm

Our algorithm is based on two observations:

1. **The number of possible patterns is small.** We observe that if there are only a few S-boxes in each round (i.e., if t is small), then even for a relatively large number r of rounds, the number of possible patterns of r -round characteristics with a small number of active S-boxes is rather small. Specifically, since r rounds contain only tr S-boxes, the number of r -round patterns with at most a active S-boxes is at most $\binom{tr}{\leq a} \triangleq \sum_{i=0}^a \binom{tr}{i}$. For reasonably small values of t , r and a , this number is quite small, and we can iterate all of them. For example, for $t = 4$, $r = 9$ and $a = 4$, there are only $\binom{36}{\leq 4} \approx 2^{17}$ distinct patterns.
2. **All characteristics following a fixed pattern can be enumerated efficiently.** We observe that once we fix a pattern (i.e., fix the active and inactive S-boxes), we can typically calculate the actual characteristics that follow this pattern in an efficient way. This is the result of the fact that once the activity/inactivity of each S-box is determined, all the possible characteristics reside in a restricted linear subspace that can be easily calculated using linear algebra.

Specifically, we denote the input difference of the characteristic by 128 variables, and “linearize” the chain of intermediate encryption differences by adding 8 new variables each time 8 state bits enter an active S-box. Since the active S-boxes are the only non-linear operations in the encryption process, all intermediate differences can be described as linear combinations of at most $128 + 8a$ variables. On the other hand, each inactive S-box in the pattern restricts the intermediate difference at the input of the S-box

to zero, giving rise to 8 linear equations in the state variables. As there are at least $rt - a$ inactive S-boxes, we obtain a system containing at least $8(rt-a)$ linear equations in at most $128+8a$ variables, which can be efficiently solved.⁴ For a sufficiently small a (compared to rt , i.e., when most of the S-boxes are inactive), the expected dimension of the subspace in which the possible characteristics reside is small.

After calculating the linear subspace of all possible characteristics, we apply a post-filtering phase that enumerates the elements of the subspace, and filters out characteristics in which the active S-box transitions are impossible (according to the difference distribution table of the S-box).⁵ Given that the dimension of the subspace is small enough, we can efficiently post-filter its elements, and thus output all the possible characteristics for the given pattern.

Combining the two observations, when t, r and a are not too large, we can efficiently enumerate all the possible r -round differential characteristics with at most a active S-boxes. The analysis of the algorithm, presented in the next subsection, shows that the complexity of the algorithm is proportional to $\binom{tr}{\leq a}$, given that the output size (i.e., the number of possible characteristics) is not too large.⁶ As a result, the algorithm is practical for a surprisingly wide choice of parameters (e.g., for $t = 4$ as in Zorro, $r = 10$ rounds and at most $a = 10$ active S-boxes, its complexity is still below 2^{32}).

2.2 Detailed Description of the Algorithm

We fix the global parameters t, r, a . The algorithm iterates over the $\binom{tr}{\leq a}$ distinct differential patterns, and for each of them, applies the two-step pattern analysis algorithm described below.

Calculating the Linear Subspace of a Pattern. We maintain a symbolic representation of the 128-bit state difference at round i , ST_i , using 128 linear combinations. Each linear combination is initialized with a 1-bit variable, representing the corresponding unknown state difference bit in the first round $\Delta(X_0)$ (before the first S-box layer). Additionally, we allocate a linear equation system E_i (which is empty at first), that describes linear constraints on the characteristic, which are imposed by the inactive S-boxes. At the end of the algorithm (after the final round, r), the subspace of all the possible characteristics is described by the null-space of E_r .

⁴ Note that some of the equations may be linearly dependent; this depends on the exact structure of the linear transformation.

⁵ Note that the solution of the linear equations yields all the intermediate differences, and in particular, the input and output differences of the active S-boxes.

⁶ As we are mainly interested in characteristics with the smallest number of active S-boxes, their number is typically not very large, and thus it is reasonable to assume that the output size is small.

The following round-linearization algorithm describes how we extend ST_i and E_i to ST_{i+1} and E_{i+1} , according to the activity pattern of the S-boxes in round $i + 1$ (starting from round $i = 0$).

Extending Linearization by 1 Round

1. Allocate and initialize $ST_{i+1} \leftarrow ST_i, E_{i+1} \leftarrow E_i$.
2. For each S-box S of round i :
 - (a) If S is inactive according to the pattern of round i , add 8 equations to the system E_{i+1} , that equate the corresponding 8 bits in ST_{i+1} to zero. If the dimension of the null-space of E_{i+1} is 0 (i.e., there is no non-zero solution to the system, and thus no matching characteristic), return ST_{i+1} and E_{i+1} as $NULL$, and exit.
 - (b) If S is active according to the pattern of round i , replace the corresponding 8 linear combinations in ST_{i+1} with the newly allocated variables.
3. Set $ST_{i+1} \leftarrow L(ST_{i+1})$, i.e., update the symbolic state ST_{i+1} according to the linear function of the cipher, L .

Given a pattern, the linear subspace of all possible characteristics for r rounds is calculated with the following algorithm:

Calculate Linear Subspace

1. Initialize ST_0 with 128 new variables, and E_0 with an empty set of equations.
2. For $i = 0$ to $i = r - 1$, run the extension algorithm for round $i + 1$, calculating ST_{i+1} and E_{i+1} . If they are $NULL$, return $NULL$ and exit.
3. Output a basis B for all the possible characteristics of the pattern using the null space of E_r . This basis is represented as a set of b free (unconstrained) linear variables, and linear combinations of these variables, as follows: the 128 linear combinations of the initial state ST_0 , and the $16 \cdot a$ linear combinations of all the inputs/outputs of the a active S-box transitions (according to the pattern).

Post-Filtering the Linear Subspace of a Pattern. Once we obtain a basis B for all the possible characteristics of the pattern, we apply a simple post-filtering algorithm.

1. For each of the 2^b possible values of the free variables:
 - (a) For each active S-box transition:
 - i. Calculate the actual input/output for the S-box transition by plugging in the values of the free variables.
 - ii. Check in the difference distribution table of the cipher whether the differential transition is possible, and if not, go back to Step 1.
 - (b) Output the full characteristic according to the current value of the free variables.

We note that it is possible to optimize the post filtering in various situations by choosing the free variables to be input/output bits of a restricted set of S-boxes. This enables us to iterate in advance only over the input/output difference transitions that are possible according to the difference distribution table of these S-boxes. The optimization can be particularly useful when the filtered linear subspace is of a relatively large dimension (and thus, we have less restrictions on the choice of free variables).

Complexity Analysis. Let $T(\text{node})$ be the average complexity of evaluating a node in the recursive tree, without iterating and post-filtering the solutions. As the number of evaluated nodes is proportional to $\binom{tr}{\leq a}$, the complexity of the algorithm can be estimated by the formula $\binom{tr}{\leq a} \cdot T(\text{node}) + SOL$, where SOL is the total number of solutions that we need to post-filter.⁷ Since we cannot determine in advance the value of SOL , we will estimate it according to the total number of characteristics which remain *after* post-filtering (i.e., the actual output size), which we denote by OUT .

In order to relate SOL and OUT , we note that an arbitrary input-output transition for an S-box is possible with probability of (at least) about $2^{-1.5}$ (this is true for the Zorro S-box, and for the AES S-box, the probability is even closer to 2^{-1}), and thus if we have at most a active S-boxes, then we expect that $OUT \geq SOL \cdot 2^{-1.5a}$, or $SOL \leq OUT \cdot 2^{1.5a}$. Consequently, the time complexity of the algorithm can be upper bounded by $\binom{tr}{\leq a} \cdot T(\text{Node}) + OUT \cdot 2^{1.5a}$. Assuming that the output size OUT is not too big, the complexity of the algorithm is proportional to $\binom{tr}{\leq a}$.

2.3 Optimized Search Algorithm Using Pattern-Prefix Search

In this section we describe an optimization of the characteristic search algorithm, which is based on the observation that we can analyze together many common patterns with the same prefix. This allows us to dispose of all the patterns whose common prefix is not possible (instead of analyzing and disposing each one separately). In addition, this algorithm reduces the average amount of work (mostly linear algebra) performed for each pattern. We note that we cannot provide an improved theoretical analysis for this algorithm. However, this algorithm appears to give a significant advantage over the basic algorithm in practice.

The algorithm *PPS* (Pattern-Prefix Search) iterates over the tree of possible prefixes of patterns using the *DFS* (Depth First Search) algorithm. The global parameters of *PPS* are the number of rounds to analyze, r , the number of S-boxes in each round, t , and the maximal number of active S-boxes, a . The parameters which are passed to each node of the tree are: the round number i , the current S-box index in the round $s \in \{0, 1, \dots, t - 1\}$, the current number of active S-boxes in the prefix, ca , and ST_i , E_i (as in the standard pattern-analysis algorithm). Thus, the *PPS* algorithm is initially called with parameters

⁷ As post-filtering a solution is very simple, we assume it can be done in unit time.

$PPS(i, s, ca, ST_0, E_0)$, where $i = 0$, $s = 0$, $ca = 0$, ST_0 is initialized with 128 new variables and E_0 is an empty set of equations.

$PPS(i, s, ca, ST_i, E_i)$:

1. If $i = r$ (i.e., we finished iterating over all the S-boxes of the pattern), then the r -round pattern is fully determined by the path to the root of the tree. Thus, calculate the basis B for all the possible characteristics of the pattern (using E_r). Finally, post-filter the characteristics (as in the pattern-analysis algorithm), and return them.
2. Allocate a node n_1 for the case that S-box with index s in round i is inactive (duplicating the current ST_i , E_i): For this node, add 8 equations to the system E_i , which equate the corresponding 8 bits in ST_i to zero. Denote the (yet undetermined) output set of this node as OUT_1 .
 - If the dimension of the null-space of E_i is 0 (i.e., there is no non-zero solution to the system, and thus no matching characteristic), delete this node and set $OUT_1 = \emptyset$.
 - Otherwise, the dimension of the null-space is greater than 0. If $s = t - 1$ (i.e., we finished iterating over all the S-boxes of the current round i), then set $ST_{i+1} = L(ST_i)$ (i.e., update the symbolic state ST_{i+1} according to the linear function of the cipher, L), also set $E_{i+1} = E_i$. Recursively call $PPS(i + 1, 0, ca, ST_{i+1}, E_{i+1})$ and set OUT_1 according to the returned output.
 - Otherwise, the dimension of the null-space is greater than 0, and $s < t - 1$. Recursively call $PPS(i, s + 1, ca, ST_i, E_i)$ and set OUT_1 according to the returned output.
3. If $ca = a$ (i.e., we have reached the maximum number of active S-boxes), return OUT_1 .
4. Otherwise ($ca < a$) allocate a node n_2 for the case that S-box with index s in round i is active (duplicating the current ST_i , E_i): For this node, replace the corresponding 8 linear combinations in ST_i with newly allocated variables. Denote the (yet undetermined) output set for this node as OUT_2 .
 - If $s = t - 1$ (i.e., we finished iterating over all the S-boxes of the current round i), then set $ST_{i+1} = L(ST_i)$ and $E_{i+1} = E_i$. Recursively call $PPS(i + 1, 0, ca + 1, ST_{i+1}, E_{i+1})$ and set OUT_2 according to the returned output.
 - Otherwise, $s < t - 1$. Recursively call $PPS(i, s + 1, ca + 1, ST_i, E_i)$ and set OUT_2 according to the returned output.
5. Return $OUT_1 \cup OUT_2$.

3 Generic Key-Recovery Algorithm for Differential Attacks on PSP Networks

In this section we present a key recovery algorithm for differential attacks exploiting the small number t of S-boxes in each round of the cipher. As in Section 2, we

describe the algorithm on the example of an AES-like cipher, in which a 128-bit state is represented as a 4-by-4 matrix of bytes, and the S-boxes (which are the only non-linear operation) act on t bytes in each round. We show that given an r -round differential characteristic with probability p , one can attack $r + \lceil 16/t \rceil$ rounds (i.e., $\lceil 16/t \rceil$ rounds in addition to the characteristic) with data and time complexity of only about $2 \cdot p^{-1}$, using negligible memory. First, we present an overview of the algorithm, and then we give a more detailed description.

3.1 An Overview of the Algorithm

For sake of simplicity, we assume that t divides 16, but the algorithm can be easily adapted to any value of t . We denote the intermediate difference of the characteristic after i rounds by Δ_i , and thus the characteristic determines Δ_i for $i \in \{0, 1, \dots, r\}$. The algorithm requires the encryption of p^{-1} plaintext pairs with input difference Δ_0 , and thus we expect that at least one of them is a right pair (i.e., follows the characteristic) with high probability. However, since we only have the output after $r + 16/t$ rounds, there are no obvious filtering conditions on the ciphertext pair, and a trivial differential attack would fail to distinguish between right and wrong pairs.

In order to work around this problem, we first note that given the actual values at the output of round $r + 16/t$, there is, on average, only one 128-bit key that leads to the fixed difference of Δ_r .⁸ In this attack, we efficiently find the key suggestion (or suggestions in general) for each of the p^{-1} ciphertext pairs, and then we perform a trial encryption in order to test whether it is the correct key. Hence, we show that instead of determining the right pair, it is sufficient to efficiently attach a candidate key to each pair.

Our strategy resembles ‘‘Attack-C’’ of Albrecht and Cid [1]. In Attack-C, the adversary tests suggestions for the key, obtained by solving non-linear equations constructed using the fixed final difference of the characteristic and each of the ciphertexts pairs. In our case, we use a similar strategy, but without directly solving any non-linear equation. Instead, we use a linearization technique similar to the technique used in our search algorithm to determine the candidate key efficiently by solving two systems of linear equations.

The algorithm first ‘‘linearizes’’ the last $16/t$ rounds by expressing the output difference $\Delta_{r+16/t}$ as a linear combination of the fixed difference Δ_r and some auxiliary variables. We start with the difference Δ_r and examine its evolution through round $r + 1$. Since there are t S-boxes in round $r + 1$, after the S-box layer there are (at most) $8t$ unknown bits. Hence, we add $8t$ variables to denote this difference so that Δ_{r+1} can be expressed as a linear combination of Δ_r and these $8t$ variables.⁹ We continue through rounds $r + 2, r + 3, \dots, r + 16/t$, and

⁸ When partially decrypting the two ciphertexts through the last $16/t$ rounds until round r with a random key, their intermediate difference is equal to Δ_r with probability 2^{-128} .

⁹ Note that unlike the characteristic search algorithm, there is no need for 128 initial variables, since the ‘‘initial’’ difference Δ_r is fixed.

finally we obtain a representation of $\Delta_{r+16/t}$ as a linear combination of Δ_r and $8t \cdot (16/t) = 128$ variables. Note that this procedure does not depend on the actual ciphertexts, and can be performed during preprocessing. After obtaining the p^{-1} ciphertext pairs, we plug the output difference $\Delta_{r+16/t}$ into the system of equations, find all the 128 intermediate variables, and thus all intermediate differences $\Delta_{r+1}, \dots, \Delta_{r+16/t-1}$.

After the differential sequence is determined, we can efficiently obtain the corresponding key suggestions to test. This is due to the fact that the determined differential transitions for the $(16/t) \cdot t = 16$ S-boxes give us the actual possible transition *values* (as each input/output difference suggests on average a single actual value). Assuming that the subkeys are interleaved with the state by a XOR operation (as in most SP networks), this gives 128 linear equations in the subkey bits, which are usually sufficient to recover the key easily.¹⁰

We note that the number of additional rounds can be further increased from $16/t$ if the differential characteristic is chosen such that its output difference Δ_r forces some S-boxes in the next rounds to be inactive. In such a case, the number of auxiliary variables in the linearization stage is decreased, and thus, more rounds can be covered by 128 auxiliary variables. As will be shown in Section 5, this is the case in our attack on Zorro, where the r -round characteristic is chosen such that out of the 8 S-boxes in rounds $(r+1)$ and $(r+2)$, only four are active. As a result, rather than attacking $r+16/4 = r+4$ rounds, we are able to break $r+5$ rounds with the same complexity.

The full details of the algorithm are given below. Its data complexity is $2 \cdot p^{-1}$ chosen plaintexts and its time complexity is a bit more than $2 \cdot p^{-1}$ (and is estimated as $4 \cdot p^{-1}$), since the analysis of each encrypted pair is very efficient (it essentially involves solving two small sets of linear equations). The algorithm requires negligible memory to store two small matrices.

3.2 A Detailed Description of the Algorithm

In order to avoid abundance of variables, we assume that the number of S-boxes in each round is $t = 4$ (as in Zorro), and thus the attack targets $r+4$ rounds. The algorithm can be easily adapted to any value of t .

The Main Key-Recovery Algorithm. The algorithm makes use of two auxiliary matrices, A_1 and A_2 , that are independent of the actual key and data, and are computed during preprocessing (to be described below).

- Given the 96×128 matrix A_1 , and Δ_{r+4} , the 96-bit vector $A_1 \cdot \Delta_{r+4}$ describes all the $12 \cdot 8 = 96$ unknown output differences for the S-boxes of rounds $r+1$, $r+2$ and $r+3$. Note that once the output differences of these 12 S-boxes are known, computing the full $\Delta_{r+1}, \Delta_{r+2}$ and Δ_{r+3} can be done by simple linear algebra.

¹⁰ If the key schedule is linear (as in Zorro), this can be done instantly by solving a system of linear equations. For more complex key schedules like that of AES, the key can typically be easily recovered by a guess-and-determine procedure.

- Given the $128 \times (128 + 256)$ matrix A_2 , and a $(128 + 256)$ -bit vector v (comprised of the 128-bit ciphertext, and $2 \cdot (32 \cdot 4) = 256$ -bit input-output values of all the S-boxes of the last 4 rounds), the product $A_2 \cdot v$ gives a suggestion of the 128-bit key K .

The full algorithm is as follows:

1. Compute the matrices A_1 and A_2 (as described below).
2. Ask for the encryptions of p^{-1} plaintext pairs with input difference Δ_0 . For each pair (P, C) and (P', C') :
 - (a) Compute $\Delta_{r+4} = C \oplus C'$, and then calculate $A_1 \cdot \Delta_{r+4}$. This allows to compute the input-output differences of the 16 S-boxes in rounds $r + 1, r + 2, r + 3, r + 4$.
 - (b) Check for each of the 16 S-boxes, whether the input-output difference transitions are possible according to the difference distribution table. If any of them is impossible, discard this pair and analyze the next pair by going back to Step 2.
 - (c) Compute according to the difference distribution table, a list of vectors $List$, containing $2 \cdot (32 \cdot 4) = 256$ -bit vectors, specifying all the possible input-output values of all the 16 S-boxes of the last 4 rounds.
 - (d) For each 256-bit vector in $List$, denoted by w :
 - i. Denote by v the $(128 + 256)$ -bit vector, comprised of the 128-bit ciphertext C , and the 256-bit vector w (specifying the input-output values for all the S-boxes of the last 4 rounds). Obtain a suggestion for the key K by computing product $A_2 \cdot v$.
 - ii. Test the key using a trial encryption, and if it succeeds, return it.

Complexity Analysis. The data complexity of the attack is $2 \cdot p^{-1}$ chosen plaintexts. For each plaintext-ciphertext pair, we perform some simple linear algebra operations, whose complexity is generally proportional to a full cipher evaluation.¹¹ As noted in the beginning of this section, we expect to test only 1 key per plaintext pair, and thus we can estimate the time complexity of the attack to be slightly higher than $2 \cdot p^{-1}$ cipher evaluations (given that the preprocessing complexity is negligible compared to p^{-1}).

The memory complexity of the attack is less than 2^{10} words of 128 bits, required in order to store A_1 and A_2 . Note that the elements of $List$ can be generated “on-the-fly”, and we do not need to store them.

Calculating the Differential Transitions from the Output Difference. This preprocessing algorithm is given as input Δ_r (which is known from the

¹¹ We can further reduce the complexity of the linear algebra using various low-level techniques (e.g., by using Gray-Codes), but these are out of the scope of this paper.

characteristic) and computes the 96×128 matrix A_1 defined above. The algorithm symbolically maintains the state difference of round i (Δ_i), denoted by ST_i (which is initialized for $i = r$ with the known Δ_r).

1. For each round $i \in \{r, r+1, r+2, r+3\}$:
 - (a) Given ST_i , compute ST_{i+1} by allocating $4 \cdot 8 = 32$ new linear variables for the output of the 4 S-boxes of round $i+1$, and then symbolically applying the linear layer L , obtaining $ST_{i+1} = L(ST_i)$ (i.e., a symbolic representation of Δ_{i+1}).
2. Given the 128 computed symbolic expressions ST_{r+4} (as functions of a total of $4 \cdot 32 = 128$ linear variables), invert the 128×128 matrix. This gives a matrix which calculates the S-box output differences of rounds $r+1, r+2$ and $r+3$ (and $r+4$) as functions of Δ_{r+4} (note that we do not actually need to allocate the 32 variables for Δ_{r+4} in order to compute this matrix). Denote by A_1 the first 96 rows of this matrix (calculating the S-box output differences of rounds $r+1, r+2$ and $r+3$).

Calculating the Key From the Ciphertext and S-box Transition Values. This preprocessing algorithm computes the $128 \times (128 + 256)$ matrix A_2 defined above. The algorithm first symbolically describes all the $(32 \cdot 4) = 128$ S-box output values in the decryption process of a (symbolic) ciphertext C , as linear combinations of the 128 variables of C , the 128 variables of K , and the $(32 \cdot 4) = 128$ input values of all the intermediate S-boxes. This is done by iteratively computing the symbolic description of the values obtained in the decryption process of C through rounds $r+4, r+3, r+2, r+1$ (from the decryption side), and expressing for each round, the outputs of the S-box transitions as linear combinations of the previous variables. Finally, the algorithm performs Gaussian elimination to express the 128 variables of the key as linear combinations in terms of the other $128 + 256$ variables, giving the matrix A_2 .

As the idea of this algorithm is very similar to the one of the previous algorithm (which computes A_1), we do not give its full description in this paper.

4 Key-Recovery Algorithm for Linear Attacks on PSP Networks

In this section we present a key recovery algorithm for linear attacks exploiting the small number t of S-boxes in each round. We show that given an r -round linear characteristic with bias q , one can attack $r + \ell$ rounds (i.e., ℓ rounds in addition to the characteristic) with data complexity¹² of $c \cdot q^{-2}$, time com-

¹² The value of c is determined by the amount of recovered subkey material and the desired success rate according to the formula suggested by Selçuk in [15] or its refinements from [5]. For example, for $t = 4$ (32-bit subkey) and success rate of 84%, we need to fix $c = 3.7$. For the full 128-bit key and success rate of 78.8% we need to fix $c = 7$.

plexity of $q^{-2} + t\ell \cdot 2^{8t\ell+8}$, and memory complexity of $\min(c \cdot q^{-2}, 2^{8t\ell})$. As in the differential case, the algorithm is based on linearization of the rounds after the characteristic. An additional tool used here is a variant of the *partial sums* technique introduced by Ferguson et al. [9].

A 1-Round Attack. For sake of clarity, we first present the algorithm in the case of $\ell = 1$. Thus, we want to attack $r + 1$ rounds exploiting an r -round linear characteristic. We denote the mask of the characteristic after i rounds by Ω_i , determining Ω_i for $i \in \{0, 1, \dots, r\}$. The algorithm works by asking for the encryptions of $c \cdot q^{-2}$ arbitrary plaintexts, and thus we expect to obtain a strong linear distinguisher after r rounds.

Obviously, the naive attack (guessing the last subkey and checking whether the linear relation holds) is worse than exhaustive key search for a 128-bit cipher, since the last round subkey consists of 128 bits. A better approach is to exchange the order of operations in the final round, such that the final key addition is performed right after the S-box layer, and the final linear layer becomes the last operation in the encryption process. This can be done by replacing the final round subkey with an equivalent key.¹³ As a result, in order to compute $\Omega_r \cdot X_r$, where X_r denotes the state after round r , it is sufficient to guess only the $8t$ equivalent subkey bits that affect the S-boxes of the last round. Thus, the attack complexity is reduced to $2^{8t} \cdot q^{-2}$.

The next optimization is useful when $c \cdot q^{-2} > 2^{8t}$ (as in the case of Zorro). We write $\Omega_r = \Omega_{8t} \oplus \Omega_{128-8t}$, namely, we divide the mask Ω_r between two masks — one that affects only the $8t$ bits in the S-boxes, and all the rest (as a result $\Omega_{8t} \cdot \Omega_{128-8t} = 0$). If two “ciphertexts” (i.e., partially decrypted ciphertexts through the linear layer L , which in the case of AES is composed of MC and SR) have the same value in the bits masked by Ω_{8t} , then for *any* key guess, they yield the same value for $\Omega_{8t} \cdot X_r$. Hence, we count for each of the $8t$ bits that enter the S-box, how many times they were suggested (if $\Omega_0 \cdot P \oplus \Omega_{128-8t} \cdot X_r = 0$, we increment the counter corresponding to the $8t$ bits, and if $\Omega_0 \cdot P \oplus \Omega_{128-8t} \cdot X_r = 1$, we decrement this counter). After counting how many times an $8t$ -bit value is suggested (again, compensating for the difference in the values of $\Omega_0 \cdot P \oplus \Omega_{128-8t} \cdot X_r$), we can analyze the $8t$ -bit value itself, and just increment/decrement the observed bias by the value of its corresponding counter. The resulting attack algorithm is as follows:

1. Initialize 2^{8t} counters to zero.
2. Collect $c \cdot q^{-2}$ plaintext/ciphertext pairs (P_i, C_i) .
3. For each ciphertext C_i , compute $Z_i = L^{-1}(C_i)$.
4. For each pair (P_i, Z_i) :
 - If $\Omega_0 \cdot P \oplus \Omega_{128-8t} \cdot Z_i = 0$, increment the counter corresponding to the value of the $8t$ bits of Z_i .

¹³ This procedure is common in attacks on AES, where the equivalent key is defined by $\tilde{K} = SR^{-1}(MC^{-1}(K))$, or in our notations $\tilde{K} = L^{-1}(K)$.

- Else, decrement the counter corresponding to the value of the 8t bits of Z_i .
5. For all 8t key bits guess:
- (a) Initialize a bias counter to 0.
 - (b) For any value of the 8t bits masked by Ω_{8t} , use the guess of the key bits to evaluate $\Omega_{8t} \cdot X_r$.
 - (c) If $\Omega_{8t} \cdot X_r = 0$, add to the bias counter, the counter associated with the 8t “ciphertext” bits, otherwise, decrement by the same value.
 - (d) Output the key with the maximal bias from 0.

The advantage of this approach over the previous one is that the expensive partial decryption step is done only 2^{8t} times, rather than $c \cdot q^{-2}$ times. The time complexity of the algorithm is $c \cdot q^{-2} + 2^{2 \cdot 8t}$, and its memory complexity is $\min(c \cdot q^{-2}, 2^{8t})$.

The algorithm can be further refined by dividing the key guessing procedure into t steps using the partial-sum technique. In the first step, we guess only the 8 subkey bits corresponding to a single S-box and partially decrypt only through this S-box, summing over the relevant counters. After this step, there are only $2^{8(t-1)}$ possible values (for the $8(t-1)$ bits, as the 8 bits corresponding to the “guessed” S-box are merged into a single entry). This process can be repeated for the next 8 subkey bits, until all 8t equivalent subkey bits are guessed. As the complexity of each of these stages is 2^{8t+8} operations, the overall time complexity of the attack becomes $c \cdot q^{-2} + t2^{8t+8}$ operations. The memory complexity remains $\min(c \cdot q^{-2}, 2^{8t})$.

Finally, we note that when the key addition layer is composed of XOR, we can optimize the parity evaluations by applying the algorithm of [6]. This algorithm, based on Fast Fourier Transform, allows computing the biases of all combinations of values and keys for a single S-box in time $3 \cdot 8 \cdot 2^8 = 2^{12.6}$ rather than 2^{16} as in a straightforward implementation. Hence, the time complexity of our attack becomes $c \cdot q^{-2} + t2^{8t+4.6}$.

An ℓ -Round Attack. In order to extend the attack to $r+\ell$ rounds, we linearize the last ℓ encryption rounds. Namely, we represent the bits of the state X_r as a linear function of the ciphertext bits and $8t\ell$ auxiliary variables (similarly to the differential attack, we add 8 variables each time an active S-box is encountered). As in the case $\ell = 1$, we observe that if two partially decrypted ciphertexts agree on $8t\ell$ bits, then they agree also on $\Omega_r \cdot X_r$. Hence, we can group the ciphertexts into $2^{8t\ell}$ sets according to the values of these bits, and execute the same algorithm as in the case of $\ell = 1$.

The complexity of the attack is $D = c \cdot q^{-2}$ known plaintexts, $M = \min(c \cdot q^{-2}, 2^{8t\ell})$ 128-bit memory blocks, and $T = c \cdot q^{-2} + t\ell \cdot 2^{8t\ell+4.6}$ operations, where each operation is less than a single round decryption.

We note that the complexity of the attack can be further reduced if the linear characteristic is chosen in such a way that only t' of the active S-boxes in round $r+1$ affect the output mask $\Omega_r \cdot X_r$. In such a case, the number of sets to which we group the ciphertexts is reduced to $2^{8((\ell-1)t+t')}$, and the attack’s

complexity is reduced accordingly. As described in Appendix A, this is the case in our linear attack on Zorro, where only 2 of the 4 active S-boxes in the last round affect the output mask. This also changes the memory complexity to $M = \min(c \cdot q^{-2}, 2^{8t(\ell-1)+8t'})$.

5 Practical Cryptanalysis of the Full Zorro

In this section we apply our generic algorithms to the lightweight block cipher Zorro.

5.1 Description of Zorro

Zorro is an AES-based 128-bit lightweight block cipher proposed by Gérard et al. at CHES 2013 [10]. The cipher executes 24 AES-like rounds, where the key schedule simply adds the 128-bit master key every four rounds, as shown at the top of Figure 2.

Each Zorro round is made of four AES-like operations, namely SB^* , AC , SR and MC (see the bottom of Figure 2). SR and MC are exactly the same as the ones used in AES, whereas AC for round i adds the four constants $(i, i, i, i \ll 3)$ to the 4 bytes of the first row. The main difference of Zorro from the AES is its non-linear operation SB^* , which contains only 4 S-boxes (instead of 16), located in the first row of the state matrix. Moreover, the actual 8×8 S-box is different than the one used in AES. However, as the S-box implantation has only a limited effect on our results, we refer the interested reader to the design document [10] for its specification.

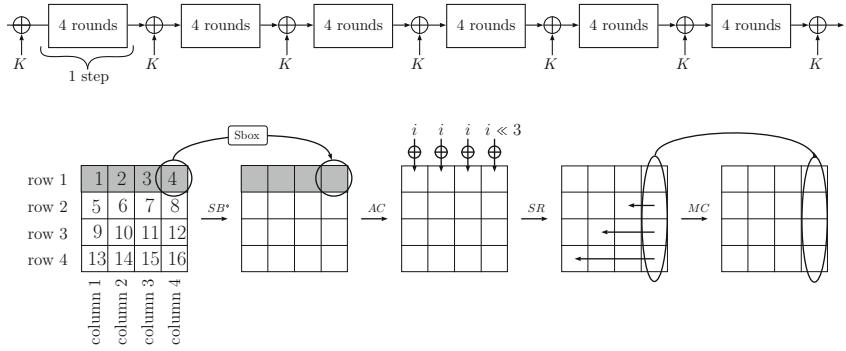


Fig. 2. The Key Schedule and Round Function of Zorro

Summary of Attacks on Zorro. Table 1 summarizes the previously published and our new attacks on full Zorro.¹⁴ We note that although the (independent and concurrent) work of Rasoolzadeh et al. [14] exploited the same characteristics as we do (that were found by them manually), their attack complexities are higher by a factor of 2^{12} , due to the use of inferior attack techniques.

Table 1. Previous, Independent and New Key-Recovery Attacks on Full Zorro

Source	Time	Data	Memory	Technique
[16]	$2^{112\ddagger\ddagger}$	2^{112} CP	negligible	Differential
[14] [†]	$\approx 2^{55}$ ^{††}	$2^{55.12}$ CP	2^{17}	Differential
[14] [†]	$2^{57.85}$	$2^{45.44}$ KP	2^{17}	Linear
Sec. 5.2	2^{45}	$2^{41.5}$ CP	2^{10}	Differential
App. A	2^{45}	2^{45} KP	2^{17}	Linear

KP - Known plaintext, CP - Chosen plaintext

[†] The results were obtained concurrently and independently of ours.

^{††} The reported time complexities of [14, 16] are lower. However, in order to calculate the time complexity, we take into account the time required for generating the data.

5.2 Differential Cryptanalysis of Full Zorro

In order to mount a differential attack on Zorro, we first apply the differential characteristic search algorithm of Section 2.2, and then use the key recovery technique of Section 3.

Differential Characteristic Search. We applied the differential search algorithm of Section 2.2 to the full Zorro. The highest probability characteristic for Zorro (for more than 7 rounds) is obtained by concatenating several instances of the 4-round iterative characteristic described¹⁵ in Figure 4 (given in the appendix). In fact, there are 5 additional linearly-dependent variants (over $GF(2^8)$) of the presented characteristic with the same probability.

Key Recovery for the Differential Attack. In order to exploit the characteristic in an attack, we extend it up to round 19 (see Figure 4). The resulting 19-round characteristic has 8 active S-boxes in total, and has probability of $(6/256)^8 \approx 2^{-43}$. We used the optimized version of our characteristic search tool (pattern-prefix search) to prove that it is the highest probability characteristic for the full 19 rounds.

¹⁴ The table does not include the results of [11], which attack a weak-key class.

¹⁵ We note that similar iterative characteristics were independently found in [14].

A straightforward application of the algorithm presented in Section 3 can be used to attack $19 + 16/4 = 23$ rounds. However, as mentioned in Section 3, more rounds can be attacked if the characteristic is chosen such that several S-boxes after the characteristic are inactive, and this is the case here. First, we observe that the state difference after 19 rounds (i.e., the output difference of the characteristic) contains 2 inactive S-boxes (see Figure 4). Furthermore, we can exploit the specific super S-box structure of Zorro (and of AES-based designs in general), and extend the characteristic with 2 additional inactive S-boxes in round 20 (see Figure 4). Thus, we have a total of 16 active S-boxes in the last 5 rounds (similarly to 4 fully active Zorro rounds), allowing to attack 5 rounds in addition to the 19 rounds of the characteristic.

According to Section 3, as the 19-round characteristic has a probability of about $p = 2^{-43}$, the data complexity of the attack is about $2 \cdot p^{-1} = 2^{44}$ chosen plaintexts, its time complexity is about 2^{45} , and its memory complexity is less than 2^{10} .

We can reduce the data complexity of the attack by a factor of 6 by using structures that exploit all the 6 characteristics of probability $p = 2^{-43}$. This is a common technique in differential cryptanalysis, and was used (for example) in [16]. Each structure we use is an affine subspace of dimension 6, which is constructed from an arbitrary plaintext, by XORing to it all the 2^6 linear combinations (over $GF(2)$) of the 6 initial differences of the characteristics of probability $p = 2^{-43}$. Thus, the data complexity is reduced by a factor of 6 to about $2^{41.5}$. The time complexity remains the same, and the memory complexity remains very small (as each structure contains only 2^6 elements).

Attack Simulation. The differential attack presented in this section was implemented and fully simulated 11 times on a single desktop PC, each simulation running for (up to) several days (the fastest took less than 8 hours, whereas the longest took about 235.5 hours). Table 2 describes the average results of the simulations, which are very close to the theoretical prediction. More detailed results are given in Table 3 (in the appendix).

Table 2. Average Simulation Results of Differential Attack on Full Zorro (Versus Theoretical Estimate)

Result	Plaintexts Encrypted	Structures Analyzed	Pairs Analyzed	Keys Suggested
Theory	$2^{41.5}$	$2^{35.5}$	2^{43}	2^{43}
Simulations (Average)	$2^{41.49}$	$2^{35.49}$	$2^{43.07}$	$2^{43.07}$

6 Design of Secure PSP Networks

In this section, we show that the weakness of Zorro is not inherently present in all PSP networks. We demonstrate this by designing a mild modification of Zorro that is *provably secure* against basic differential and linear attacks (such as those that broke the original Zorro). Finally, we discuss how to choose the parameter t (i.e., the number of S-boxes in each round) in PSP networks.

In order to quantify what we consider to be a “good” PSP network with respect to resistance against basic differential and linear attacks, we estimate the minimal number a of active S-boxes in an r -round characteristic for a very strong PSP network. Our model constructs an idealized PSP network by choosing the layer of each round uniformly at random from the space of invertible linear mappings, and it is therefore expected to provide very fast diffusion.

As described in Section 2, an r -round characteristic with a active S-boxes gives rise to a system of $8(tr - a)$ linear equations in $8(16 + a)$ variables (using the notations of Section 2). Based on our randomness assumption, we expect a solution when $8(16 + a) \geq 8(tr - a)$, or equivalently, $a \geq (t \cdot r - 16)/2$. Namely, an r -round characteristic for an idealized PSP network is expected to have at least $(t \cdot r - 16)/2$ active S-boxes.¹⁶ We note that this inequality is somewhat oversimplified, as it does not take into account the fact that we have many possible patterns, whereas we are looking for only one valid characteristic. On the other hand, depending on the actual S-box, not all solutions are valid for a given cipher. As these two considerations have opposite effects on a , and their total effect seems relatively small for large values of r , we consider the formula $a \geq (t \cdot r - 16)/2$ to be a reasonable measure for a “good” PSP Network. As an extreme case, consider AES for which $t = 16$. Plugging $r = 4$ into the formula, we estimate that 4-round AES can be designed to have at least $a \geq (16 \cdot 4 - 16)/2 = 24$ active S-boxes in any characteristic. Indeed, it is known that the minimal number of active S-boxes in a 4-round characteristic of AES is 25 (see [7]), and thus our estimate is very close in this case.

6.1 Analysis of a Concrete PSP Network

We now construct a PSP Network which (roughly) satisfies the formula $a \geq (t \cdot r - 16)/2$ for large values of r , thus providing significantly better resistance against basic differential and linear attacks compared to Zorro. According to the full version of this paper [2], in order to avoid the weakness of Zorro, our scheme has to deviate from the AES-based design strategy. More specifically, this appendix shows that any AES-based PSP network (with small t) is likely to have 4-round iterative characteristics with a high probably. The reason for the inherent weakness of AES-based PSP networks is subtle and is detailed in

¹⁶ This formula is somewhat more conservative (from the point of view of the designer) compared to the one obtained in [10], that seems to underestimate the number of degrees of freedom available in the construction of the characteristic, thus obtaining larger values of a .

the full version of this paper [2]. Very roughly, this weakness stems from the combination of the two properties below:

1. Any MDS circulant MixColumn matrix, MC, raised to the power of 4 (i.e., $(MC)^4$) has a large space of eigenvectors (“almost fixed-points”) that satisfy $MC^4(x) = \alpha x$ for an appropriately chosen eigenvalue scalar α .
2. The order of ShiftRows is 4 (i.e., $(SR)^4$ is the identity).

Therefore, in order to avoid the high probability 4-round iterative characteristics of the type shown in Figure 4, our scheme has to deviate from the AES design strategy by changing at least one of the two properties above. In our tweaked scheme, we slightly change the ShiftRows operation such that its order is greater than 4, as described below. Furthermore, in order to avoid additional types of iterative characteristics (namely, characteristics presented in [16], which are independent of ShiftRows), we also change the locations of the S-boxes, and place them on the diagonal instead of the first row.

The modified variant of ShiftRows (denoted as SR^*) is described in Figure 3 and works as follows: The action of SR^* on rows 1,3 and 4 is the same as in the original ShiftRows. On the other hand, only the first 3 bytes of row 2 are cyclically rotated by 1 (whereas the 4'th byte remains unchanged at its position), and it is easy to see that the order of SR^* is $3 \cdot 4 = 12$. We note that this modified variant provides slightly weaker local diffusion compared to AES-based designs. However, we now show that globally, this modification significantly strengthens the resistance of the scheme against standard differential and linear attacks.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
7	5	6	8
11	12	9	10
16	13	14	15

Fig. 3. Modified ShiftRows

We first consider 11 rounds of the tweaked scheme and estimate its strength in our ideal model by plugging $t = 4$ and $r = 11$ into the formula $a \geq (t \cdot r - 16)/2$, obtaining $a \geq (4 \cdot 11 - 16)/2 = 14$. However, a more careful analysis reveals that there are many possible 11-round patterns with 13 active S-boxes ($\binom{11 \cdot 4}{13} > 2^{35}$), each giving rise to a system with $8 \cdot (16 + 13) = 8 \cdot 29$ variables and $8 \cdot (44 - 13) = 8 \cdot 31$ equations, which has a solution with non-negligible probability of $2^{8 \cdot (29 - 31)} = 2^{-16}$. Therefore, 13-round characteristics can also be expected, slightly deviating from the generic formula when we do not consider post-filtering according to the cipher's S-box.

Using the characteristic search tool presented in Section 2, we were able to prove that there exists no characteristic (or linear mask) with at most 12 active S-boxes (regardless of the cipher’s specific S-box). Considering 13 active S-boxes, there exist only a few dozens of possible characteristics for the cipher. Consequently, the behavior of the 11-round scheme closely matches our ideal one, and we conclude that it has no particular weakness against standard differential and linear cryptanalysis.

A more comprehensive differential analysis¹⁷ of the 11-round cipher reveals that none of the characteristics with 13 active S-boxes satisfies the restrictions imposed by the 13 transitions through the Zorro S-box. This is expected, as the number of possible characteristics is small, and implies that our generic formula $a \geq (t \cdot r - 16)/2$ predicted the exact value of $a = 14$ in this case. Indeed, we were able to find about 2^{32} 11-round characteristics with 14 active S-boxes using our tool. When considering the specific Zorro S-box for post-filtering these solutions, about 2^{22} valid 14-round differential characteristics remain. The highest-probability differential characteristic with 14 active S-boxes is described in Figure 5 (in the appendix), having probability of about $2^{-86.8}$. Since the highest differential transition probability for the Zorro S-box is about $2^{-4.7}$, this proves that the best 11-round characteristic has probability of at most $\min(2^{-86.8}, 2^{15} \cdot (-4.7)) = 2^{-70.5}$. Consequently, the best characteristic for 22 rounds of the cipher has probability of at most $2^{-70.5 \cdot 2} = 2^{-141}$ (note that for the stronger AES S-box, the bound is even lower).

For 12 rounds of the cipher, we were able to prove that there exists no characteristic (or mask) with at most 14 active S-boxes. However, we did not run the tool for more than 14 active S-boxes, as this is too time-consuming for a standard desktop PC. Of course, it would be interesting to further optimize the search tool and efficiently analyze more rounds.¹⁸ Nevertheless, even in their current state, our results are sufficient for demonstrating that the security of our modified Zorro variant with respect to standard differential and linear attacks is close to that of an idealized PSP network with the same parameters. Indeed, according to the formula $a \geq (t \cdot r - 16)/2$, the bound of at least $a = 30$ active S-boxes (or a differential characteristic probability bound of 2^{-141}) should be obtained for $r = 19$ rounds. Thus, the gap between the expected behavior of the scheme and what we can prove, is only 3 rounds for a probability bound as low as 2^{-141} , and if we consider the stronger AES S-box (for which the probability bound of 2^{-141} can be obtained with only 24 active S-boxes), this gap is even smaller.

¹⁷ The linear analysis is very similar, and we omit it from this paper.

¹⁸ One could try to incorporate dynamic programming techniques into our tool, similarly to the algorithms of [3,4]. However, this seems far from straightforward, as concatenating two patterns requires the relatively complex operation of intersecting their corresponding linear subspaces.

6.2 How to Choose the Number of S-boxes in Each Round in PSP Networks?

We now use the insight gained in this paper to revisit one of the main questions posed in [10], namely: for a PSP network, what is the value of t that offers security with the minimal number of S-boxes? In [10], it was concluded that for 128-bit ciphers with 8-bit S-boxes, the optimal value is $t = 4$, when considering security against standard differential and linear cryptanalysis. However, our formula $a \geq (t \cdot r - 16)/2$ shows that the total number of S-boxes in the scheme, $t \cdot r$, required to guarantee that a of them are active (and thus to obtain a bound on the characteristic probability) is fixed to $(2 \cdot a) + 16$, regardless of the value of t . Furthermore, according to Sections 3 and 4, the number of S-boxes that need to be added at the end of the cipher is fixed as well (e.g., to about 16 for differential attacks), and is independent of t .

Since it is possible to use $t = 16$ as in AES, this seems to question the effectiveness of PSP networks in thwarting side-channel attacks via masking techniques. Indeed, when considering resistance against standard differential and linear cryptanalysis, it seems that there is no gain in using partial non-linear layers. However, we still claim that the combination of partial non-linear layers with strong linear layers has an advantage, when taking into consideration other types of attacks.

In order to demonstrate this potential advantage, we consider AES-128, where 4 rounds are sufficient for assuring that any characteristic has probability lower than 2^{-128} . Despite its strength against differential and linear cryptanalysis, 4-round AES-128 is an extremely weak cipher due to strong structural properties, and can be broken in 2^{10} chosen plaintexts and time (see [9]). In fact, as the best attacks on AES-128 can break 7 rounds (e.g., see [8]), one has to (at least) double the number of rounds to 8 (and double the number of S-boxes to $16 \cdot 8 = 128$) in order to obtain a secure cipher.

PSP networks, on the other hand, employ many strong linear layers through more rounds, and thus seem to better mitigate structural attacks. Consequently, one could build a secure PSP network where the number of S-boxes is closer (compared to AES) to the bound $(2 \cdot a) + 16$.

Finally, we note that (generalized) Feistel structures employ only partial linear layers, and therefore may require many more than $(2 \cdot a) + 16$ S-boxes to resist standard differential and linear cryptanalysis. Furthermore, due to the partial linear layers, some Feistel structures are particularly vulnerable to structural attacks such as impossible differential cryptanalysis.

We conclude that PSP networks may allow for more efficient masking techniques to mitigate side-channel attacks. However, the optimal choice of the number of S-boxes in each round has to be made for each specific design separately, after evaluating its security against a broad class of attacks, which are out of the context of this paper.

7 Conclusions

In this paper, we introduced new algorithms for differential and linear cryptanalysis of PSP networks. Using these algorithms, we were able to devise and fully simulate a practical attack against the block cipher Zorro. We then closely examined PSP networks, and concluded that they should not be based directly on the AES design strategy. Finally, we designed and analyzed a tweak of Zorro and used it to show that PSP networks do not have an inherent flaw. We do not formally propose to use this tweak, as this would require defining its concrete number of rounds and performing full analysis against many types of known attacks. Nevertheless, we believe that our tweak may provide a good starting point for building future designs. Alternatively, one can think of building PSP networks based on bit-oriented design strategies, such as the one used for the block cipher Serpent. Regardless of concrete design strategies, we believe that the tools developed in this paper will be useful in future PSP network design and analysis.

A Details of the Linear Cryptanalysis of Full Zorro

In this section, we describe our linear attack on full Zorro.

Linear Characteristic Search. We applied the linear characteristic search algorithm of Section 2 to 23-round Zorro. Similarly to the differential case, the best characteristics are concatenations of 4-round iterative linear characteristics. These characteristics can be viewed as counterparts of the differential ones, and follow a similar representation as in Figure 4. The resulting 23-round linear characteristic has 10 active S-boxes, and thus has a bias of $q = (56/256)^{10} \approx 2^{-22}$. As in the differential case, we used our characteristic search tool to prove that it is the best linear characteristic for 23 rounds.

Key Recovery for the Linear Attack. Using the algorithm of Section 4, we can attack $23 + 1 = 24$ rounds with data complexity of 2^{44} known plaintexts, time complexity of $2^{44} + 2^{2 \cdot 8+9} \approx 2^{44}$ encryptions, and memory complexity of $\min(2^{44}, 2^{2 \cdot 8+1}) = 2^{17}$ 32-bit words.¹⁹ The attack recovers 2 bytes of equivalent key $\tilde{K} = SR^{-1}(MC^{-1}(K))$ (i.e., the two bytes used in the “active” S-boxes in round 24), which is the result of exchanging the order of the final key addition and linear operations SR and MC .

In order to recover additional 2 bytes of \tilde{K} , we can simultaneously and independently (using the same data) exploit the variant of the same linear characteristic, in which the 2 columns are swapped. Furthermore, we can simultaneously exploit another variant of the iterative characteristic which spans rounds 2–24

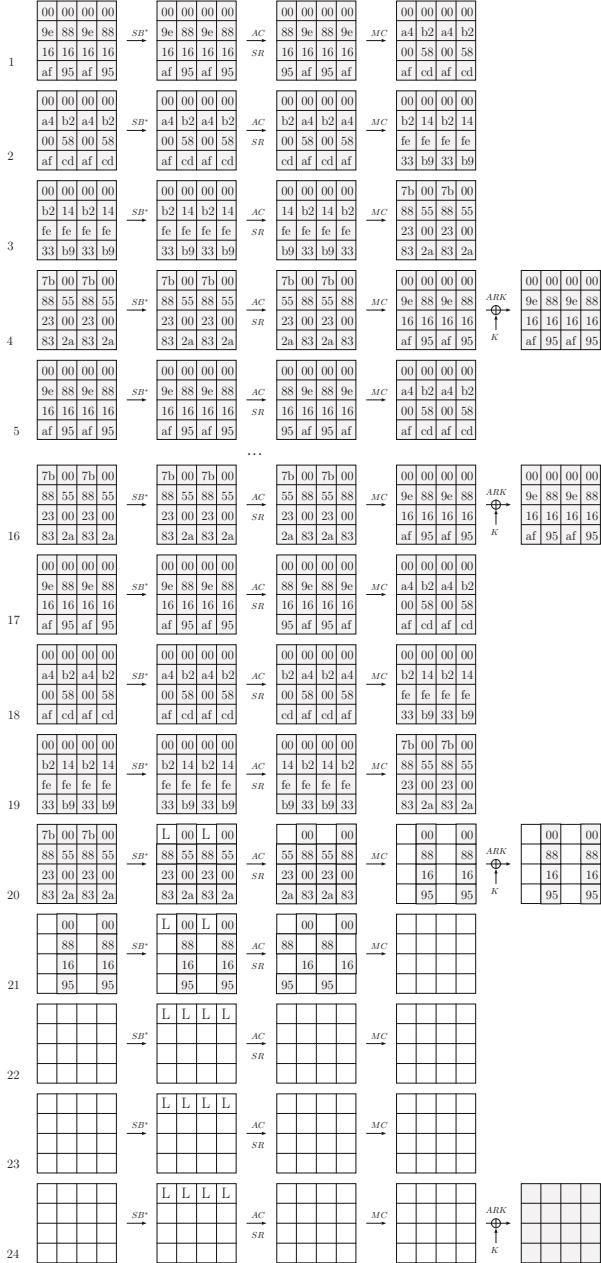
¹⁹ Note that since there are only two active S-boxes in round 24 that affect the output bias of the characteristic, the memory complexity is 2^{17} , and not 2^{33} as it would be in the worst case.

(with the active S-boxes in round 2), and apply the key recovery on the encryption side. This allows us to recover 2 bytes of K , and additional 2 bytes can be simultaneously recovered by swapping the columns in the last characteristic. As the time complexity bottleneck in all of these 4 simultaneous attacks is the actual collection of data, the total time complexity of recovering the 8 bytes of key material remains about 2^{44} , and the memory complexity is less than $4 \cdot 2^{(2 \cdot 8)+1} = 2^{19}$ words of 32 bits, or 2^{17} words of 128 bits.

After determining the 8 bytes of key material used in rounds 1 and 24 which contribute to all non-linear operations, we can “peel off” this non-linearity and apply the same ideas to the inner rounds 2 and 23 in order to recover the 8 additional (linear combinations of) key bytes, which contribute to the non-linearity in these rounds. This is done by exploiting the iterative characteristics in which the active S-boxes are in round 2, and in round 23. However, due to the dependency of the inner-round attacks on the previously recovered 8 bytes, it is not obvious how to perform these attacks simultaneously, and thus (in order to avoid the large memory overhead of storing the original data) we can request additional 2^{44} known plaintexts in order to recover the rest of the key. This leads to an attack that uses 2^{45} known plaintexts, runs in 2^{45} time, and requires memory of about 2^{17} words of 128 bits.

Table 3. Simulation Results of Differential Attack on Full Zorro (Versus Theoretical Estimate)

Simulation	Plaintexts Encrypted	Structures Analyzed	Pairs Analyzed	Keys Suggested
Theory	$2^{41.5}$	$2^{35.5}$	2^{43}	2^{43}
Simulation Average	$2^{41.49}$	$2^{35.49}$	$2^{43.07}$	$2^{43.07}$
1	$2^{38.30}$	$2^{32.30}$	$2^{39.89}$	$2^{39.84}$
2	$2^{38.50}$	$2^{32.50}$	$2^{40.08}$	$2^{40.06}$
3	$2^{38.56}$	$2^{32.56}$	$2^{40.14}$	$2^{40.10}$
4	$2^{38.77}$	$2^{32.77}$	$2^{40.35}$	$2^{40.34}$
5	$2^{38.86}$	$2^{32.86}$	$2^{40.44}$	$2^{40.44}$
6	$2^{39.12}$	$2^{33.12}$	$2^{40.70}$	$2^{40.69}$
7	$2^{40.59}$	$2^{34.59}$	$2^{42.17}$	$2^{42.22}$
8	$2^{40.83}$	$2^{34.83}$	$2^{42.41}$	$2^{42.43}$
9	$2^{42.90}$	$2^{36.90}$	$2^{44.49}$	$2^{44.47}$
10	$2^{43.07}$	$2^{37.07}$	$2^{44.66}$	$2^{44.67}$
11	$2^{43.21}$	$2^{37.21}$	$2^{44.79}$	$2^{44.79}$



The output differences of the 16 S-boxes marked with L are initially unknown. They are linearized and recovered according to Section 3, leading to an efficient key recovery.

Fig. 4. Differential Attack on Full Zorro

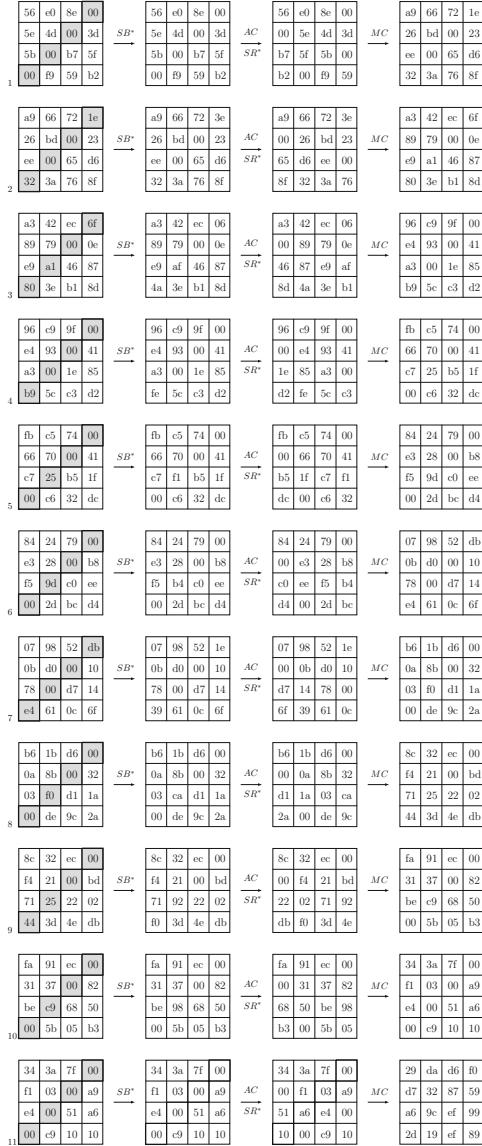


Fig. 5. Best 11-round Characteristic with 14 Active S-boxes for the Tweaked PSP Network

Acknowledgments. The research of the first author was partially supported by the Israeli Ministry of Science, Technology and Space, and by the Check Point Institute for Information Security.

References

1. Albrecht, M., Cid, C.: Algebraic Techniques in Differential Cryptanalysis. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 193–208. Springer, Heidelberg (2009)
2. Bar-On, A., Dinur, I., Dunkelman, O., Lallemand, V., Keller, N., Tsaban, B.: Cryptanalysis of SP Networks with Partial Non-Linear Layers. Cryptology ePrint Archive, Report 2014/228 (2014). <http://eprint.iacr.org/>
3. Biryukov, A., Nikolić, I.: Automatic Search for Related-key Differential Characteristics in Byte-oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 322–344. Springer, Heidelberg (2010)
4. Biryukov, A., Nikolić, I.: Search for Related-Key Differential Characteristics in DES-Like Ciphers. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 18–34. Springer, Heidelberg (2011). http://dx.doi.org/10.1007/978-3-642-21702-9_2
5. Bogdanov, A., Tischhauser, E.: On the Wrong Key Randomisation and Key Equivalence Hypotheses in Matsui's Algorithm 2. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 19–38. Springer, Heidelberg (2014)
6. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Improving the Time Complexity of Matsui's Linear Cryptanalysis. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 77–88. Springer, Heidelberg (2007)
7. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002). <http://dx.doi.org/10.1007/978-3-662-04722-4>
8. Derbez, P., Fouque, P.-A., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 371–387. Springer, Heidelberg (2013). http://dx.doi.org/10.1007/978-3-642-38348-9_23
9. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.L.: Improved Cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
10. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.-X.: Block Ciphers that Are Easier to Mask: How Far Can We Go? In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 383–399. Springer, Heidelberg (2013)
11. Guo, J., Nikolic, I., Peyrin, T., Wang, L.: Cryptanalysis of Zorro. Cryptology ePrint Archive, Report 2013/713 (2013). <http://eprint.iacr.org/>
12. Matsui, M.: On Correlation Between the Order of S-boxes and the Strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 366–375. Springer, Heidelberg (1995). <http://dx.doi.org/10.1007/BFb0053451>
13. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In: Wu, C.-K., Yung, M., Lin, D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer, Heidelberg (2012). http://dx.doi.org/10.1007/978-3-642-34704-7_5
14. Rasoolzadeh, S., Ahmadian, Z., Salmasizadeh, M., Aref, M.R.: Total Break of Zorro using Linear and Differential Attacks. Cryptology ePrint Archive, Report 2014/220 (2014). <http://eprint.iacr.org/>
15. Selçuk, A.A.: On Probability of Success in Linear and Differential Cryptanalysis. *J. Cryptology* **21**(1), 131–147 (2008)
16. Wang, Y., Wu, W., Guo, Z., Yu, X.: Differential Cryptanalysis and Linear Distinguisher of Full-Round Zorro. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 308–323. Springer, Heidelberg (2014). http://dx.doi.org/10.1007/978-3-319-07536-5_19

Hash Functions

The Sum Can Be Weaker Than Each Part

Gaëtan Leurent¹(✉) and Lei Wang²

¹ Inria, Paris, France

gaetan.leurent@inria.fr

² Nanyang Technological University, Singapore, Singapore

wang.lei@ntu.edu.sg

Abstract. In this paper we study the security of summing the outputs of two *independent* hash functions, in an effort to increase the security of the resulting design, or to hedge against the failure of one of the hash functions. The exclusive-or (XOR) combiner $H_1(M) \oplus H_2(M)$ is one of the two most classical combiners, together with the concatenation combiner $H_1(M) \parallel H_2(M)$. While the security of the concatenation of two hash functions is well understood since Joux’s seminal work on multi-collisions, the security of the sum of two hash functions has been much less studied. The XOR combiner is well known as a good PRF and MAC combiner, and is used in practice in TLS versions 1.0 and 1.1. In a hash function setting, Hoch and Shamir have shown that if the compression functions are modeled as random oracles, or even *weak* random oracles (*i.e.* they can easily be inverted – in particular H_1 and H_2 offer no security), $H_1 \oplus H_2$ is indifferentiable from a random oracle up to the birthday bound.

In this work, we focus on the preimage resistance of the sum of two narrow-pipe n -bit hash functions, following the Merkle-Damgård or HAIFA structure (the internal state size and the output size are both n bits). We show a rather surprising result: the sum of two such hash functions, e.g. SHA-512 \oplus Whirlpool, can never provide n -bit security for preimage resistance. More precisely, we present a generic preimage attack with a complexity of $\tilde{O}(2^{5n/6})$. While it is already known that the XOR combiner is not preserving for preimage resistance (*i.e.* there might be *some* instantiations where the hash functions are secure but the sum is not), our result is much stronger: for *any* narrow-pipe functions, the sum is not preimage resistant.

Besides, we also provide concrete preimage attacks on the XOR combiner (and the concatenation combiner) when one or both of the compression functions are weak; this complements Hoch and Shamir’s proof by showing its tightness for preimage resistance.

Of independent interests, one of our main technical contributions is a novel structure to control *simultaneously* the behavior of independent hash computations which share the same input message. We hope that breaking the pairwise relationship between their internal states will have applications in related settings.

Keywords: Hash functions · Combiners · XOR combiner · Preimage attack

1 Introduction

Hash functions are a very important class of primitive in modern cryptography, used in almost every secure system. A hash function $H : \{0, 1\}^* \mapsto \{0, 1\}^n$ takes an arbitrary length input and produces an n -bit output or digest. Hash functions are used in many setting with various security requirements; the general expectation is that a hash function should behave like a random function from $\{0, 1\}^*$ to $\{0, 1\}^n$. More concretely, the main security notions expected from a hash function are:

Collision resistance. It should be hard to find two messages $M \neq M'$ with $H(M) = H(M')$.

Second-preimage resistance. Given a message M , it should be hard to find $M' \neq M$ with $H(M) = H(M')$.

Preimage resistance. Given a target hash value \bar{H} , it should be hard to find M with $H(M) = \bar{H}$.

Since generic collision attacks require $2^{n/2}$ work, and generic preimage attacks require 2^n work, a secure hash function should have the same level of resistance.

In order to build more secure hash functions, or to protect oneself against future cryptanalysis advances, such as the devastating attacks of Wang *et al.* against the SHA family [36, 37], a practical countermeasure might be to combine two different hash functions. The goal is that the combined hash function can only be broken when both components are weak. In particular, this reasoning was used by the designers of SSL [11] and TLS [5], who combined MD5 and SHA-1 in various ways. More precisely, the Key Derivation Function of TLS v1.0/v1.1 uses a sum of HMAC-MD5 and HMAC-SHA-1.¹ The designers explain [5]: “In order to make the PRF as secure as possible, it uses two hash algorithms in a way which should guarantee its security if either algorithm remains secure.”

There are two classical hash function combiners: the concatenation combiner $H_1(M)\|H_2(M)$ and the XOR combiner $H_1(M)\oplus H_2(M)$. In a seminal work [17], Joux showed that the concatenation combiner with narrow-pipe hash functions offers much less security than could be expected: it has $2n$ bits of output, but essentially offers the same security as an n -bit hash function. In this work, we carry a similar analysis for the XOR combiner. Previous work has shown that it is indistinguishable from a random oracle up to the birthday bound [14], even if the initial functions are weak; in particular, it has optimal collision resistance of $n/2$ bits. However, we show that the preimage security is much less than one might expect, with a generic preimage attack with complexity $\tilde{O}(2^{5n/6})$.

Since the goal of a combiner is to keep some security even if one of the functions is found to be weak, it is natural that the two hash functions H_1 and H_2 are independent in practice. Throughout this paper the two hash functions

¹ We note that this MD5/SHA-1 combiner has been replaced by primitives based on single hash function (e.g., SHA-256) since TLS v1.2 [6].

used in a combiner are always assumed to be independent without specifying it explicitly².

Iterated Hash Function. In this paper we consider iterated hash functions, following the Merkle-Damgård construction [4, 27] or the more general HAIFA construction [2], as shown in Figure 1. We focus on narrow-pipe designs, *i.e.* we assume that the internal state size is the same as the output size n . The message M is first split into blocks m_0, \dots, m_ℓ , and the hash function iterates a series of compression functions h_i over an internal state x , with the initial value denoted as IV . Finally, the hash value is computed with a finalization function g :

$$x_0 = IV \quad x_{i+1} = h_i(x_i, m_i) \quad H(M) = g(x_{\ell+1}, |M|)$$

In the following, we assume that the compression function is the same at every step ($\forall i, h_i = h$) in order to simplify the notations, but it is straightforward to apply the attack with the corresponding function at each step.

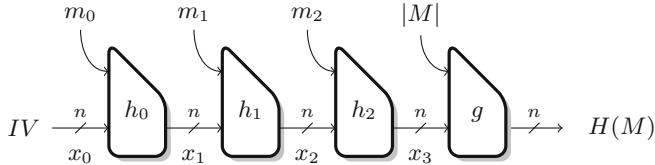


Fig. 1. Iterated hash function

1.1 Related Works

Combiners have been studied in several different settings. For generic attacks, the compression functions are modeled as random functions, in order to devise attacks that don't use any weakness of the compression functions. On the other hand, some work assumes that the compression function is a *weak* random oracle (that can easily be inverted), and prove that some constructions are still secure in this model. Finally, more theoretical work focus on the notion of *robustness*, *i.e.* the existence of a reduction from attacks on the hash functions to attacks on the combiner.

² If the two hash functions can be related, it is trivial that the XOR combiner is not security-preserving. For instance, let $H_2(M) := H_1(M) \oplus \text{const}$, where const is a constant. If H_1 is ideally secure, then H_2 is also ideally secure. However, the XOR combiner $H_1(M) \oplus H_2(M) = \text{const}$ for any message M .

Analysis of the Concatenation Combiner. The concatenation combiner $H_1(M) \parallel H_2(M)$ is probably the most studied one. In 2004, Joux [17] described surprising attacks on the concatenation of two narrow-pipe hash functions using multicollisions: while the output size is $2n$ bits, the concatenation can at most provide $n/2$ -bit security for collision resistance and n -bit security for preimage resistance³. In particular, the concatenation is not security-amplifying. On the other hand, the concatenation combiner is robust for preimages and collisions, which gives a matching lower bound for generic attacks.

Later, Hoch and Shamir [14] evaluated the security of the concatenation combiner with two weak hash functions. More precisely, the two hash functions are narrow-pipe Merkle-Damgård, and the compression functions are modeled as weak random oracles (as defined by Liskov [21]), *i.e.*, the adversary is given additional interfaces to receive (random) preimages of the compression functions. They have proven that in this model, the concatenation combiner is still indifferentiable from a random oracle with $n/2$ -bit security, implying (at least) the same security bound for collision resistance and preimage resistance. The bound is matched by Joux's attack for collisions, but there is a gap with Joux's attack for preimages, with complexity 2^n , which might be interesting to investigate further.

Mendel *et al.* analyzed some dedicated instantiations of the concatenation combiner [25], in particular using the hash function MD5. We omit the details and refer interested readers to [25].

Analysis of the XOR Combiner. The XOR combiner has received less analysis. The work of Hoch and Shamir [14] actually proves the security of the XOR combiner as an intermediate result: it is also indifferentiable from a random oracle up to $2^{n/2}$ queries in the weak random oracle model. In particular, this proves that there are no generic attacks with complexity smaller than $2^{n/2}$. For collision resistance, the bound is tight, since it is matched with the generic birthday attack bound. On the other hand, for preimage resistance, there exists a gap between the $n/2$ -bit proven bound and the n -bit expected ideal security bound.

To the best of our knowledge, no preimage attacks have been shown against the XOR combiner. Therefore, the preimage security of the XOR combiner against generic attacks is still an open problem, and will be the main topic of our work. We will also consider the preimage security of the XOR combiner with weak hash functions, and study the tightness of Hoch and Shamir's bound.

Robust Combiners. In the last years, the general problem of combining two (or more) hash functions H_1 and H_2 has been extensively studied from a theoretical point of view. These works focus on the notion of a *robust* combiner: a robust combiner is secure with respect to property α as long as one of the underlying hash functions is secure for α . It can be shown that the concatenation combiner is a robust combiner for collision-resistant hash functions and for

³ The attacks actually require only one of the functions to be narrow-pipe.

MACs, while the XOR combiner is robust for PRFs and for MACs [20]. More advanced combiners have been constructed in order to be robust for multiple properties simultaneously [8–10]. The notion was mostly studied via the black-box reduction model.

A series of results have showed that robust combiners for collision resistance and preimage resistance cannot have an output length significantly shorter than the sum of the output length of the underlying hash functions [3, 28, 32, 33]. Since the XOR combiner is length preserving, this shows that it is not robust for collision resistance and preimage resistance.

Actually, the impossibility results are in part due to the stringent requirement from the black-box reduction model. In order to overcome this limitation, Mittelbach introduced the *idealized* random oracle model [29]. He gives a construction of a short output combiner with optimal collision and preimage security in this model⁴ (assuming that one of the functions is ideal): Cryptophia’s short combiner uses the sum of two hash functions with some pre-processing of the messages (to allow non-independent functions).

More generally, we point out that a combiner being non-robust does not necessarily mean there is an attack. The non-robustness results only show that the security of the combiner cannot be proved with a *reduction* from the security of the hash functions. In particular, the XOR combiner is not robust for collision-resistance, or even collision-resistance preserving. However, Hoch and Shamir’s work proves that there are no generic collision attacks on this construction, either with ideal compression function, or even with weak compression functions. This arguably makes the XOR a useful combiner for collision resistance. Regarding preimage security, the non-robustness result does not imply that the XOR of two concrete hash functions is weak, and the simplicity and short output of this construction still make it attractive.

1.2 Our Results

In this work, we study the preimage security of the XOR combiner, the main remaining open problem for classical combiners. We show that, surprisingly, the sum of two narrow-pipe n -bit hash functions can never achieve n -bit security for preimage resistance. More precisely, we find a generic preimage attack with a complexity of $\tilde{O}(2^{5n/6})$. It does not exploit any structural weakness of the compression functions and hence is applicable even if the compression functions are two ideal random oracles. Thus, even if the two hash functions are n -bit secure for preimage resistance, the XOR combiner is at most $5n/6$ -bit secure for preimage resistance. In other words, *the sum can be weaker than each part*.

The attack is based on a novel technique to break the pairwise relationship between the internal states of the two hash functions. More precisely, the two hash functions H_1 and H_2 share the same input message, and hence the internal states of their iterative compression function computations are related. We

⁴ A mistake in the initial proof and construction was later fixed by Mennink and Preneel [26].

control the computation chains of H_1 and H_2 simultaneously by constructing a new message structure \mathcal{M} , and two sets of internal states \mathcal{A} for H_1 and \mathcal{B} for H_2 such that: for any value A from \mathcal{A} and any value B from \mathcal{B} , we can derive a message $M_{A,B}$ from \mathcal{M} such that $H_1(M_{A,B})$ produces A and $H_2(M_{A,B})$ produces B . Hence we can select states from \mathcal{A} and \mathcal{B} independently. After that, we use a birthday match to find a message block m , a value A from \mathcal{A} and a value B from \mathcal{B} such that $h_1(A, m) \oplus h_2(B, m)$ is equal to the target hash digest, where h_1 and h_2 are the compression functions of H_1 and H_2 respectively. Finally we derive the message $M_{A,B}$ from \mathcal{M} , and output $M_{A,B} \parallel m$ as a preimage of the target hash digest.

Our preimage attack is also applicable to Cryptophia's short combiner [26, 29]. This construction has been proven to provide optimal collision and preimage resistance, assuming that at least one of the initial functions is a monolithic random oracle, but our attack does not violate the security proof, because we use the fact that both functions have an iterative structures with an n -bit internal state. Still, this shows that with many practical hash functions, the combiner will be weaker than the initial functions. Our results also show that the XOR combiner and Cryptophia's combiner are not robust in the semi-black-box model introduced by Mittelbach [29]⁵, even with independent hash functions H_1 and H_2 .

Our analysis on the XOR combiner is also interesting for dedicated hash function design. The hash function family RIPEMD [7] is based on a compression function with two parallel lanes, added together at the end of each compression function. Interestingly, RIPEMD-160 has been quite resilient to cryptanalysis [22–24, 34], and are still considered secure. Several more recent designs use parallel lanes in a similar way (combining them at the end of each compression function call), such as HAS-V [31], FORK [15] and LANE [16]. It might be tempting to use parallel lanes during the full iteration, and to combine them only at the end. Indeed, the designers of SHA-V [12] used this approach: the 160-bit version of SHA-V has two parallel lanes, combined at the end with a modular sum. This is equivalent to summing two different hash functions, and hence our attack can be applied to SHA-V.

Another contribution of this paper is to present concrete preimage attacks on the XOR combiner with one or both weak hash functions (defined in [21]). The complexity of our attacks is $\tilde{O}(2^{n/2})$. Furthermore, the attack can be extended to the concatenation combiner with two weak hash functions under the same complexity. It can be seen that these attacks match the bound of Hoch and Shamir's security proof [14], and hence fulfill the gaps pointed out in Section 1.1. It implies the tightness of Hoch and Shamir's proof on the classical combiners with weak hash functions for preimage resistance.

Finally, we would like to highlight the technical interests of this paper. We devise a novel structure named interchange structure to simultaneously control two (or more) hash lanes with the same input message, and succeed in further

⁵ Loosely speaking, a combiner is robust with respect to property α if it is (at least) as secure as the stronger underlying hash function for α .

relaxing the pairwise relation between the internal states of lanes. It is indeed a step of technical advance compared with previous extensive studies on this topic, and hence will hopefully have applications or lead to new technical development in related settings. We refer to the open discussions in Section 7 for more details.

1.3 Notations and Roadmap in the Rest of Paper

We use the following notations:

H_1, H_2 : hash functions

IV_1, IV_2 : initial values for H_1 and H_2 , respectively

h_1, h_2 : compression functions of H_1 and H_2 , respectively

h_1^*, h_2^* : compression functions iterated over several blocks
(in particular, $H_i(\mathbf{M}) = h_i^*(IV_i, \mathbf{M})$)

m : message block

M : message chunk ($n/2$ blocks)

\mathbf{M} : long message (several chunks)

a_j, b_k : chains for H_1 and H_2 , respectively

a_j denotes a generic chain, while a_{j_0} denotes a particular chain

A_j, B_k : end points of the chains

n : hash function output size

Roadmap. Section 2 provides an overview of our generic preimage attack on the XOR combiner. Sections 3, 4, and 5 elaborate the attack procedure step by step in details. Section 6 presents the applications and extensions of the attack. Finally we conclude the paper and discuss future directions in Section 7.

2 Overview of the Attack

We first give an overview of the techniques and the structures used in the attack, while more detailed descriptions will be given in the following sections.

The main idea is to consider several chains of internal states reached by processing a common message \mathbf{M} from different starting points (note that the message \mathbf{M} is not fixed in advance, but will be determined when building the structure). More precisely, the message \mathbf{M} is denoted as the *primary* message, and divided in several chunks: $\mathbf{M} = M_0 \parallel M_1 \parallel \dots$ (as discussed later, a chunk will consist of several message blocks). We denote chains of internal states for H_1 as a_j , and the individual states of the chain as a_j^i , with $h_1^*(a_j^i, M_i) = a_j^{i+1}$. Similarly, we denote chains for H_2 as b_k , with $h_2^*(b_k^i, M_i) = b_k^{i+1}$. When considering both hash functions, message block M_i leads from the pair of states (a_j^i, b_k^i) to (a_j^{i+1}, b_k^{i+1}) , which is denoted:

$$(a_j^i, b_k^i) \xrightarrow{M_i} (a_j^{i+1}, b_k^{i+1}).$$

Switch structure. Next we build special structures called *switches* in order to jump between chains in a controlled way. A switch allows to jump from a specific pair of chains (a_{j_0}, b_{k_0}) to a different pair of chains (a_{j_0}, b_{k_1}) using a secondary message chunk M'_i , in addition to the normal transitions using chunk M_i of the primary message \mathbf{M} :

$$(a_j^i, b_k^i) \xrightarrow{M_i} (a_j^{i+1}, b_k^{i+1}) : \quad \text{normal transition for each chain}$$

$$(a_{j_0}^i, b_{k_0}^i) \xrightarrow{M'_i} (a_{j_0}^{i+1}, b_{k_1}^{i+1}) : \quad \text{jump from chains } (a_{j_0}, b_{k_0}) \text{ to } (a_{j_0}, b_{k_1})$$

In order to simplify the notations, we often omit the chunk index, in order to show only the chains that are affected by the switch.

The main message chunk M_i and the secondary message chunk M'_i are determined when building the switch, and the main message defines the next state of all the chains. We note that the secondary message chunk M'_i should only be used when the state is $(a_{j_0}^i, b_{k_0}^i)$. A simple example is depicted in Figure 2.

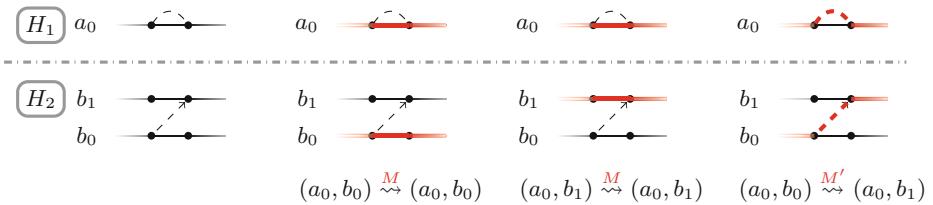


Fig. 2. A single switch: jump from (a_0, b_0) to (a_0, b_1) by using M' (dashed lines) instead of M (solid lines)

Alternatively, a switch can be designed to jump from (a_{j_0}, b_{k_0}) to (a_{j_1}, b_{k_0}) . We defer the details of the construction to Section 3; it can be built with a complexity of $\tilde{O}(2^{n/2})$.

Interchange Structure. By combining several simple switches, we can build an interchange structure with starting points IV_1 and IV_2 and ending points $\{A_j, j = 0 \dots 2^t - 1\}$ and $\{B_k, k = 0 \dots 2^t - 1\}$, so that we can select a message ending in any state (A_j, B_k) . Figure 3 shows one possible way to build such a structure, and Figure 4 shows how to select a given message in the structure. An interchange structure with 2^t chains for each function requires about 2^{2t} switches. Since we can build a switch for a cost of $\tilde{O}(2^{n/2})$, the total structure is built with $\tilde{O}(2^{2t+n/2})$ operations.

Preimage Search. Finally, we can use an interchange structure with ending points $\{A_j, j = 0 \dots 2^t - 1\}$ and $\{B_k, k = 0 \dots 2^t - 1\}$, to build a preimage

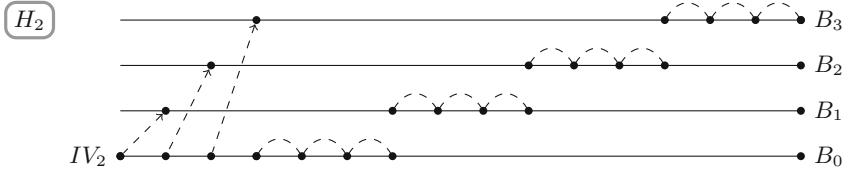
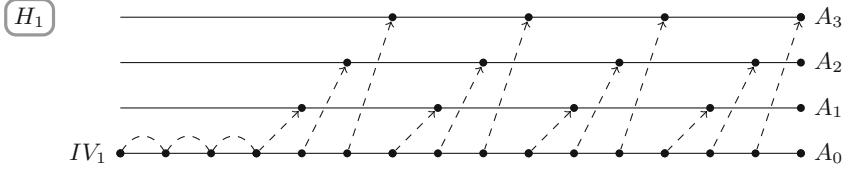


Fig. 3. Overview of an interchange structure

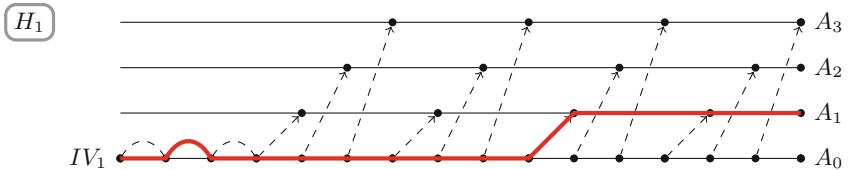


Fig. 4. Using of the interchange structure to reach output (A_1, B_2)

attack as follows. Let \bar{H} denote the target value. We select a random message block m , and we compute two lists by evaluating the compression functions after the interchange structure: $\{A'_j = h_1(A_j, m), j = 0 \dots 2^t - 1\}$ and $\{B'_k = \bar{H} \oplus h_2(B_k, m), k = 0 \dots 2^t - 1\}$. We expect a match between the lists with probability 2^{2t-n} . After about 2^{n-2t} random choices of m , we get a match (j^*, k^*) :

$$h_1(A_{j^*}, m) = \bar{H} \oplus h_2(B_{k^*}, m) \quad i.e. \quad h_1(A_{j^*}, m) \oplus h_2(B_{k^*}, m) = \bar{H}.$$

Therefore, we can construct a preimage of \bar{H} by concatenating the message leading to (A_{j^*}, B_{k^*}) in the interchange structure, and the block m (we ignore the finalization function in this section).

The complexity of the preimage search is about 2^{n-t} evaluations of the compression function, using an interchange structure with 2^t end-points.

Complexity Analysis. Building the interchange structures requires about $2^{2t+n/2}$ evaluations of the compression function, while the preimage search requires about 2^{n-t} . The optimal complexity is reached when both steps take the same time, *i.e.* $t = n/6$. This gives a complexity of $\tilde{O}(2^{5n/6})$.

3 The Switch Structure

We now explain how to build the switch structure at the core of our attack. This construction is strongly based on the multicollision technique of Joux [17].

Given states $a_{j_0}^i, b_{k_0}^i$ and $b_{k_1}^i$, we want to build message chunks M_i and M'_i in order to have the following transitions:

$$(a_{j_0}^i, b_{k_0}^i) \xrightarrow{M_i} (a_{j_0}^{i+1}, b_{k_0}^{i+1}) \quad (a_{j_0}^i, b_{k_1}^i) \xrightarrow{M_i} (a_{j_0}^{i+1}, b_{k_1}^{i+1}) \quad (a_{j_0}^i, b_{k_0}^i) \xrightarrow{M'_i} (a_{j_0}^{i+1}, b_{k_1}^{i+1}).$$

The main message chunk M_i is used to define the next state of all the remaining chains, while the secondary message chunk M'_i will be used to jump from chains $(a_{j_0}^i, b_{k_0}^i)$ to $(a_{j_0}^i, b_{k_1}^i)$. We note that M'_i will only be used when the state is $(a_{j_0}^i, b_{k_0}^i)$. In particular, M_i and M'_i must satisfy:

$$\begin{aligned} a_{j_0}^{i+1} &= h_1^*(a_{j_0}^i, M_i) = h_1^*(a_{j_0}^i, M'_i) \\ b_{k_1}^{i+1} &= h_2^*(b_{k_1}^i, M_i) = h_2^*(b_{k_0}^i, M'_i) \\ b_{k_0}^{i+1} &= h_2^*(b_{k_0}^i, M_i) \neq b_{k_1}^{i+1} \end{aligned}$$

We first build a multicollision for h_1^* , starting from state $a_{j_0}^i$, *i.e.* a large set \mathcal{M} of $2^{n/2}$ messages that all reach the same state $a_{j_0}^{i+1}$ ($\forall M \in \mathcal{M}, h_1^*(a_{j_0}^i, M) = a_{j_0}^{i+1}$). As shown by Joux, this can be done efficiently by sequentially building $n/2$ collisions.

Next, we evaluate $h_2^*(b_{k_0}^i, M)$ and $h_2^*(b_{k_1}^i, M)$ for all the messages M in the set \mathcal{M} . With high probability there is match between the sets of values⁶. We denote the colliding messages as M_i and M'_i , so that we have $h_2^*(b_{k_0}^i, M'_i) = h_2^*(b_{k_1}^i, M_i)$.

Finally we compute the missing chains using the message m_i : $a_j^{i+1} = h_1^*(a_j^i, m_i)$, $b_k^{i+1} = h_2^*(b_k^i, m_i)$. With high probability all the chains reach distinct values; if this is not the case, we restart the construction with a new multicollision. The full algorithm is shown as Algorithm 1, and illustrated by Figure 5; it requires about $n/2 \cdot 2^{n/2}$ evaluations of the compression functions.

4 The Interchange Structure

Let us now describe the combination of switch structures into an interchange structure. The goal of this structure is to select the final value of the H_1 computation and the H_2 computation independently. More precisely, the structure defines two sets of final values A_j and B_k , and a set of messages \mathbf{M}_{jk} such that:

$$(IV_1, IV_2) \xrightarrow{\mathbf{M}_{jk}} (A_j, B_k).$$

⁶ If this is not the case, we build a new multicollision.

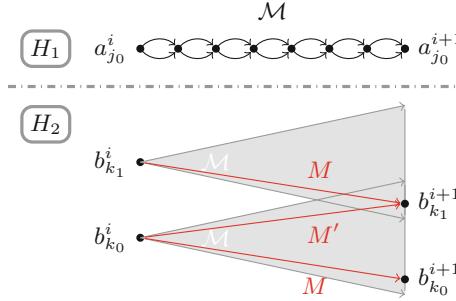


Fig. 5. Building a switch structure. First, M and M' are selected from \mathcal{M} to generate a collision (defining the new b_{k_1}), then b_{k_0} is evaluated using M .

In order to build this structure, we initialize the first chains with $a_0^0 = IV_1$, $b_0^0 = IV_2$, and set the other starting points randomly. Then, we use switches to jump for an already reachable pair (a_{j_0}, b_{k_0}) to a different pair (a_{j_0}, b_{k_1}) (or to (a_{j_1}, b_{k_0}) , respectively). By using $2^{2t} - 1$ switches, we can make all pairs reachable. There are many way to combine the switches; a simple one can be described as follow:

1. first, build switches from (a_0, b_0) to each of the (a_0, b_k) 's;
2. then for each k , build a series of switches from (a_0, b_k) to all the (a_j, b_k) 's.

In order to reach the chains (a_j, b_k) , one would activate the k -th switch in the first part to jump from (a_0, b_0) to (a_0, b_k) , and then the j -th switch in the k -th series of the second part to jump from (a_0, b_k) to (a_j, b_k) . This structure is shown in Figure 3 and a pseudo-code description is given by Algorithm 2, where the INTERCHANGE functions builds the structure, and the SELECTMESSAGE function extracts the message reaching (a_j, b_k) .

The structure can be somewhat optimized using the fact that the extra chains have no prespecified initial values. We show how to take advantage of this in Appendix A, using multicollision structures in addition to the switch structures. However, this doesn't change significantly the complexity: we need $(2^t - 1)(2^t - 1)$ switches instead of $2^{2t} - 1$. In total, we need about $n/2 \cdot 2^{2t+n/2}$ evaluations of the compression functions to build a 2^t -interchange structure.

We believe that a 2^t -interchange structure based on switches will need at least $\Theta(2^{2t})$ switches, because every switch can only increase the number of reachable pairs (a_j, b_k) by one. As shown in Appendix A some switches can be saved in the beginning but it seems that new ideas would be needed to reduce the total complexity below $\Theta(2^{2t+n/2})$.

5 Preimage Attack

Finally, we describe the full preimage attack. We first build an interchange structure with 2^t chains for each of H_1 and H_2 . We denote the ending points as

Algorithm 1. Building a single switch

```

function SWITCH( $h_1, h_2, a, b, b'$ )
   $x \leftarrow a$ 
   $\mathcal{M} \leftarrow \emptyset$ 
  for  $0 \leq i < n/2$  do
     $(m, m') \leftarrow \text{COLLISION}(h_1, x)$ 
     $\mathcal{M} \leftarrow (\mathcal{M} \parallel m) \cup (\mathcal{M} \parallel m')$ 
     $x \leftarrow h_1(x, m)$ 
  end for
   $\mathcal{H} \leftarrow \{\}$ 
  for  $M \in \mathcal{M}$  do
     $y \leftarrow h_2^*(b, M)$ 
     $\mathcal{H}[y] \leftarrow M$ 
  end for
  for  $M \in \mathcal{M}$  do
     $y \leftarrow h_2^*(b', M)$ 
    if  $\mathcal{H}[y]$  exists then
      return  $M, \mathcal{H}[y]$ 
    end if
  end for
end function

function COLLISION( $h, x$ )
   $\mathcal{H} \leftarrow \{\}$ 
  loop
     $m \leftarrow \$$ 
     $y \leftarrow h_1(x, m)$ 
    if  $\mathcal{H}[y]$  exists then
      return  $m, \mathcal{H}[y]$ 
    else
       $\mathcal{H}[y] \leftarrow m$ 
    end if
  end loop
end function

```

$\{A_j, j = 0 \dots 2^t - 1\}$ and $\{B_k, k = 0 \dots 2^t - 1\}$, and we know how to select a message \mathbf{M}_{jk} to reach any state (A_j, B_k) . When adding a message block m to one of the messages \mathbf{M}_{jk} in the interchange structure, the output of the combiner can be written as:

$$H_1(\mathbf{M}_{jk} \parallel m) \oplus H_2(\mathbf{M}_{jk} \parallel m) = g_1(h_1(A_j, m), \ell + 1) \oplus g_2(h_2(B_k, m), \ell + 1),$$

where g_1 and g_2 are the finalization functions of H_1 and H_2 , respectively, and ℓ is the length of the messages in the structure.

In order to reach a target value \bar{H} , we select a random block m , and we evaluate $\{A'_j = g_1(h_1(A_j, m), \ell + 1), j = 0 \dots 2^t - 1\}$ and $\{B'_k = \bar{H} \oplus g_2(h_2(B_k, m), \ell + 1), k = 0 \dots 2^t - 1\}$.

Algorithm 2. Building and using a T -interchange structure

```

function INTERCHANGE( $h_1, h_2, IV_1, IV_2$ )
     $a_0 \leftarrow IV_1, b_0 \leftarrow IV_2$ 
    for  $1 \leq k < T$  do
         $a_k \leftarrow \$, b_k \leftarrow \$$ 
    end for
    for  $1 \leq j < T$  do
         $(M, M') \leftarrow \text{SWITCH}(h_1, h_2, a_0, b_0, b_j)$ 
         $M \leftarrow M \parallel M; M' \leftarrow M' \parallel M'$ 
        for  $0 \leq k < T$  do
             $a_k \leftarrow h_1^*(a_k, M)$ 
             $b_k \leftarrow h_2^*(b_k, M)$ 
        end for
    end for
    for  $1 \leq j < T$  do
        for  $1 \leq i < T$  do
             $(M, M') \leftarrow \text{SWITCH}(h_2, h_1, b_j, a_0, a_i)$ 
             $M \leftarrow M \parallel M; M' \leftarrow M' \parallel M'$ 
            for  $0 \leq k < T$  do
                 $a_k \leftarrow h_1^*(a_k, M)$ 
                 $b_k \leftarrow h_2^*(b_k, M)$ 
            end for
        end for
    end for
    return  $(M, M')$ 
end function

function SELECTMESSAGE( $M, M', j, k$ )
     $\mu \leftarrow M$ 
    if  $k \neq 0$  then
         $\mu[k - 1] \leftarrow M'[k - 1]$ 
    end if
    if  $j \neq 0$  then
         $\mu[(k + 1) \cdot (T - 1) + j - 1] \leftarrow M'[(k + 1) \cdot (T - 1) + j - 1]$ 
    end if
    return  $\mu$ 
end function

```

$\ell + 1), k = 0 \dots 2^t - 1\}$. If there is a match (j^*, k^*) between the two lists, we have:

$$\begin{aligned}
A'_{j^*} = B'_{k^*} &\Leftrightarrow g_1(h_1(A_{j^*}, m), \ell + 1) = \overline{H} \oplus g_2(h_2(B_{k^*}, m), \ell + 1) \\
&\Leftrightarrow H_1(\mathbf{M}_{jk} \parallel m) \oplus H_2(\mathbf{M}_{jk} \parallel m) = \overline{H}.
\end{aligned}$$

For a random choice of m , we expect that a match exists with probability 2^{2t-n} , and testing it requires about 2^t operations⁷. We will have to repeat this procedure 2^{n-2t} times on average, therefore the total cost of the preimage search is about 2^{n-t} evaluations of h_1 and h_2 .

As explained in the previous section, building a 2^t -interchange structure requires about $n/2 \cdot 2^{2t+n/2}$ operations. Using $t = n/6$ we balance the two steps of the attack, and reach the optimal complexity of about $n/2 \cdot 2^{5n/6}$ operations for the preimage attack.

5.1 Message Length and Memory Complexity

The attack uses messages of length $n/2 \cdot 2^{2t}$, and the memory complexity of the attack⁸ is also $n/2 \cdot 2^{2t}$. The optimal choice $t = n/6$ gives messages of length $n/2 \cdot 2^{n/3}$. The memory requirement is probably not an issue⁹ for an attacker that can spend time $2^{5n/6}$, but the message length can be a problem with some hash functions that don't accept long inputs. For instance SHA-256 is only defined for message with less than 2^{64} bits (*i.e.* 2^{55} blocks).

In this case, one can apply the attack with a smaller value of t : this reduces the length of the messages, at the cost of more time spent in the preimage search step. For instance, we can mount a preimage attack against SHA-256 \oplus BLAKE-256 with complexity 2^{232} using $t = 24$, while the optimal attack with $n = 256$ would cost only $2^{220.3}$. Similarly, our attack applied to SHA-512 \oplus Whirlpool has a complexity of 2^{461} , rather than $2^{434.7}$.

6 Applications and Extensions

The attack works identically if the hash functions use the HAIFA mode rather than the plain Merkle-Damgård iteration. Also it can easily be extended to $H_1(M) \odot H_2(M)$ where \odot denotes an easy to invert group operation (for instance, a modular addition rather than the exclusive or). The attack can also be extended to hash functions H_1 and/or H_2 using an internal check-sum, such as the GOST family of hash functions, using pairs of blocks with a constant sum.

6.1 Application to the Sum of Wide-Pipe Hash Functions

The attack can also be used when the internal state size ℓ is larger than the output size n . The complexity of building a 2^t -interchange structure is related to ℓ as $\ell/2 \cdot 2^{2t+\ell/2}$. On the other hand, the complexity of the meet-in-the-middle preimage search is related to n as 2^{n-t} . The optimal complexity is $\ell/2 \cdot 2^{2n/3+\ell/6}$ by matching the two complexities with $t = n/3 - \ell/6$. Therefore our attack can be applied as long as $\ell + 6 \log(\ell) \leq 2n$ holds. For instance, we can compute preimages of SHA-224 \oplus BLAKE-224 with complexity roughly 2^{199} .

⁷ It takes $O(t \cdot 2^t)$ operations by sorting the lists, but only $2 \cdot 2^t$ using a hash table.

⁸ We only need to store the messages M and M'

⁹ For instance, the attack is on the verge of practicality with $n = 64$; the time complexity is $2^{58.3}$ and the memory complexity is $2^{26.3}$.

6.2 Application to Cryptophia's Short Combiner

Our attack can also be applied to Cryptophia's short combiner, as proposed by Mittelbach [29], and to the revised version of Mennink and Preneel [26]. This combiner computes the sum of two hash functions with some pre-processing of the message, to allow non-independent functions:

$$\begin{aligned} C(M) &= H_1(\tilde{m}_1^1 \parallel \dots \parallel \tilde{m}_\ell^1) \oplus H_2(\tilde{m}_1^2 \parallel \dots \parallel \tilde{m}_\ell^2) \\ \tilde{m}_j^1 &= H_1(0 \parallel l_1 \parallel m_j \oplus k_1) \oplus H_2(0 \parallel l_2 \parallel m_j \oplus k_2) \\ \tilde{m}_j^2 &= H_1(1 \parallel l_1 \parallel m_j \oplus k_1) \oplus H_2(1 \parallel l_2 \parallel m_j \oplus k_2) \end{aligned}$$

where k_1, k_2, l_1, l_2 is a randomly chosen key. The security proof in the ideal model shows that C is optimally preimage resistant if at least one of the hash functions is ideal.

However, if both H_1 and H_2 are narrow-pipe, we can apply our preimage attack with complexity $\tilde{O}(2^{5n/6})$. This does not violate the security proof because we need both functions to be narrow-pipe, hence not n -bit ideal¹⁰. From a practical point of view, though, it shows that in many cases (*e.g.* using SHA-512 and Whirlpool) the combiner is *weaker* than the initial functions.

6.3 Improvements Using Weaknesses of the Hash Functions

If H_1 or H_2 has known cryptographic weaknesses, more efficient attacks are possible. More precisely, if the compression function of one of the hash functions can be inverted¹¹ in time 2^t , then we can find a preimage of $H_1 \oplus H_2$ with complexity only $\tilde{O}(2^{(n+t)/2})$.

The attack is presented using the case, where (at least) one compression function is modeled as a weak compression function defined in [21], as an example. Without loss of the generality, we assume the compression function of H_2 , $h_2(x, y) = z$, is such a weak compression function, which is a random oracle with two additional interfaces as below.

- *Backward interface.* On a query (x, y, z) , it returns either a value x uniformly chosen from all the values satisfying $h_2(x, y) = z$, or \perp if no such x exists.
- *Bridging interface.* On a query (x, y, z) , it return either a value y uniformly chosen from all the values satisfying $h_2(x, y) = z$, or \perp if no such y exists.

Note that the inversion of compression function h_2 takes unit time and hence the attack against $H_1 \oplus H_2$ takes time $\tilde{O}(2^{n/2})$. The procedure is detailed as follows, which is also illustrated in Figure 6.

Let the target hash digest be denoted as \overline{H} . We firstly build an n -block long multicollision on H_1 following Joux's approach [17]. Let the final output be denoted as Z . It contains a set of up to 2^n messages that link IV_1 to Z on H_1 .

¹⁰ A large multi-collisions can be built with a cost of roughly $2^{n/2}$ in a narrow-pipe function, but costs almost 2^n for an ideal hash function.

¹¹ finding an input chaining value from the output chaining value and the message

We split every n -block long multicolliding message into halves, and collect the first half $n/2$ -block long messages as a set \mathcal{M}_1 and the second half as another set \mathcal{M}_2 . Hence for any message $M_1 \in \mathcal{M}_1$ and any $M_2 \in \mathcal{M}_2$, the concatenated message $M_1 \| M_2$ links IV_1 to Z on H_1 . Secondly, for the messages $M_2 \in \mathcal{M}_2$, we use the additional backward interface of h_2 to carry out backward computations from $\overline{H} \oplus Z$, and get values X such that $h_2^*(X, M_2) = \overline{H} \oplus Z$. We store (X, M_2) in a table T_X . On average T_X contains $2^{n/2}$ elements.¹² Finally, for each message $M_1 \in \mathcal{M}_1$, we iteratively compute h_2 forward from IV_2 , and get an internal state Y . We match Y to the elements in T_X . A match implies that concatenated $M_1 \| M_2$ links IV_2 to $\overline{H} \oplus Z$, and in turn is a preimage of $H_1(M_1 \| M_2) \oplus H_2(M_1 \| M_2) = Z \oplus \overline{H} \oplus Z = \overline{H}$. The success probability of finding such a match is not negligible since there are $2^{n/2}$ X 's and Y 's. The complexity is around $n \cdot 2^{n/2}$ that is $\tilde{O}(2^{n/2})$ by ignoring the polynomial factors.

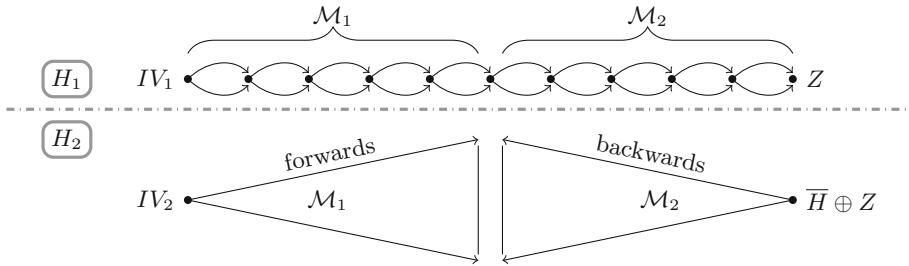


Fig. 6. Preimage attack on the XOR combiner with a weak H_2

Moreover, the above preimage attack can be extended to the concatenation combiner $H_1 \| H_2$ if both H_1 and H_2 are weak by a minor modification. This shows that the proof of Hoch and Shamir [14] is tight for preimage resistance. Here we mainly highlight the modifications. Let the target hash digest be $\overline{H}_1 \| \overline{H}_2$, where \overline{H}_1 is from H_1 and \overline{H}_2 from H_2 . After we build the multicollision on H_1 and let the output internal state be denoted as Z , we link Z to \overline{H}_1 by using the bridging interface of h_1 to receive a message m such that $h_1(Z, m) = \overline{H}_1$. This gives us a set of messages linking IV_1 to \overline{H}_1 on H_1 . Also note that for the backward computations on H_2 , the starting value should be \overline{H}_2 . The rest of the attack procedure remains the same. Hence it is easy to get that the complexity is also $\tilde{O}(2^{n/2})$.

6.4 Extension to the Sum of Three or More Hash Functions

The attack can be extended to the sum of three or more hash functions. In order to attack the sum of k functions, two different strategies are possible: either we

¹² The backward interface may output \perp for some message block. To compensate it, for the other message blocks, we make multiple queries, since they may have more than one preimages. On average, the backward interface should produce one preimage for each query.

use a simpler structure that only gives two degrees of freedom, and fixes $k - 2$ functions to a constant value, or we build an interchange structure to control all the k functions independently.

Controlling only Two Functions. The easiest way to extend the attack is to use a single chain in the $k - 2$ extra hash functions. The procedure to build a switch is modified in order to use multicollisions for $k - 1$ functions instead a simple multicollisions for one function; this costs $O(n^{k-1} \cdot 2^{n/2})$ using Joux's method [17].

As in the basic attack, we need $O(t^2)$ switches to generate a 2^t -interchange for two functions, and the preimage search costs $O(2^{n-t})$; the optimal complexity is therefore $O(n^{k-1} \cdot 2^{5n/6})$ with $t = n/6$.

Controlling all the Functions. Alternatively, we can build a more complex interchange structure in order to control all the functions independently. When attacking three functions, we will use the switch structure to jump from chains $(a_{j_0}, b_{k_0}, c_{l_0})$ to $(a_{j_0}, b_{k_0}, c_{l_1})$ (or $(a_{j_0}, b_{k_1}, c_{l_0})$ or $(a_{j_1}, b_{k_0}, c_{l_0})$, respectively). We need $2^{3t} - 1$ switches in the interchange structure to reach all the 2^{3t} triplets of chains (a switch makes only one new triplet reachable). Each switch is built using a $2^{n/2}$ -multicollision on two functions, which can be built for a cost of $O(n^2 \cdot 2^{n/2})$ following Joux's technique [17]. Therefore we can build a 2^t -interchange for a cost of $O(n^2 \cdot 2^{3t+n/2})$. More generally, for the sum of k hash functions, we can build an interchange structure for k functions for a cost of $O(n^{k-1} \cdot 2^{kt+n/2})$.

In the preimage search phase, we generate k lists of size 2^t , and we need to detect efficiently whether we can combine them to generate a zero sum. This problem can be solved using a algorithm similar to Wagner's generalized birthday algorithm [35]. If $k = 2^\kappa$, we find a solution with probability $O(2^{n-(\kappa+1) \cdot t})$ for a cost of $O(k \cdot 2^t)$. Therefore the preimage search costs $O(k \cdot 2^{n-\kappa t})$. With $k = 4$ (i.e. $\kappa = 2$), this yields a complexity of $O(n^3 \cdot 2^{5n/6})$. However, this approach is less efficient than the previous one for $k = 3$ and for $k > 4$.

To summarize, attacking the sum of k hash functions ($k \geq 2$) costs $O(n^{k-1} \cdot 2^{5n/6})$. Controlling chains independently in more than two hash function might be useful for further work, but it doesn't improve the preimage attack on the sum of k hash functions.

7 Conclusion and Open Discussions

In this work, we gave the first generic attack on the XOR combiner. Our result is rather surprising: the sum of two ideal narrow-pipe hash functions only has about $5n/6$ bits of security against preimage attacks. In particular, the *sum is easier to break* than the initial functions. Since most practical hash functions are narrow-pipe (e.g. SHA-1, SHA-256, SHA-512, Whirlpool, RIPEMD, GOST,

BLAKE, Skein...), the XOR combiner will usually provide a weaker security than the component hash functions.

Moreover, we would like to discuss a few directions for future work.

On Controlling Multiple Hash Lanes. Since 2004, several generic attacks have been found against narrow-pipe hash functions, such as the multicollision attack[17], the long-message second preimage attack[19] and the herding attack[18]. There has been extensive work to extend these attacks to more complex constructions with several computation chains, such as the concatenation $H_1(M) \parallel H_2(M)$ and the cascade $H_2(H_1(M), M)$.

As in our present work, the essential difficulty in those attacks comes from the fact that several lanes of computation share the same input message, and hence their outputs are related. If an adversary considers a naïve set of messages, the set of outputs gives random pairs of n -bit values $\{(A_i, B_i) : i \in \mathcal{I}\}$: selecting a value for the first entry of the pair gives a single candidate for the second entry, and the adversary is essentially working with a $2n$ -bit state. Previous works [1, 13, 30] have developed various message structures (mostly based on multicollision and diamond structures), in order to relax this relation. They mainly result in a set of messages \mathcal{M} such that the corresponding outputs are in a more structured set $\{(A, B_i) : i \in \mathcal{I}\}$: the first entry is a constant value A , but several options B_i can be selected for the second entry. For any value (A, B_i) , it is then possible to select a message in \mathcal{M} so that the first lane reaches A , while the second lane reaches B_i . This allows to modify the value of the second lane without affecting the first lane.

Our result is quite stronger: with the interchange structure we have a set of message such that the corresponding outputs are a set $\{(A_i, B_j) : i \in \mathcal{I}, j \in \mathcal{J}\}$, where *both* lanes have several options that can be selected independently. We hope that our technique will have applications or lead to new technical development in related settings, e.g., the open problem of generic second preimage attacks (with long messages) on the concatenation hash or on the Zipper hash [21].

On Extending to Practical Hash Function. Several practical hash functions such as RIPEMD [7] and HAS-V [31] are based on a compression function with more than one independent lanes, which interacts with each other at the end of each compression function call. It is very interesting to investigate if our attack can be further modified to attack these hash functions in future. Again the obstacle comes from the relations between internal states of lanes. Particularly, the internal states of the lanes interact with each other, which makes the relation even tighter and in turn harder to attack.

Acknowledgments. The authors would like to thank the anonymous referees for their helpful comments. Lei Wang is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

A Optimized Interchange Structure

We now describe an optimized attack using only $(2^t - 1)(2^t - 1)$ switches rather than $2^{2t} - 1$. The attack also requires multicollision structures, as introduced by Joux[17].

We replace the first $2^t - 1$ switches with a 2^t -multicollision in H_1 , and we use those messages to initialize all the b_k chains in H_2 . We can also optimize the first series of switches in H_2 in the same way: we build a 2^t -multicollision in H_2 starting from b_0 , and we use those messages to initialize the a_j chains in H_1 . This is illustrated by Figure 7, and the detailed attack is given as Algorithm 3.

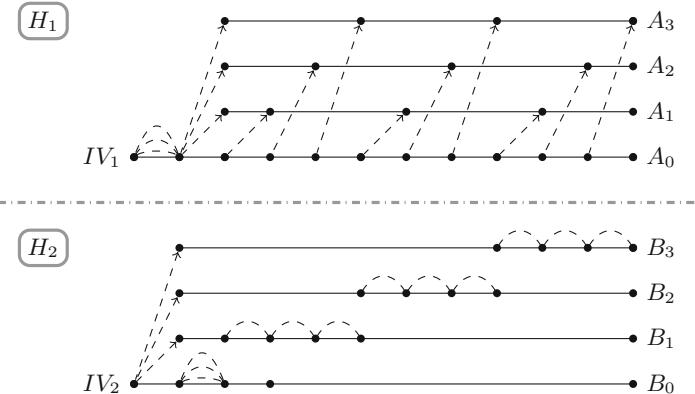


Fig. 7. Optimized interchange structure

Algorithm 3. Optimized T -interchange structure

```

function INTERCHANGE( $h_1, h_2, IV_1, IV_2, T$ )
   $a_0 \leftarrow IV_1, b_0 \leftarrow IV_2$ 
   $\mathcal{M}_0 \leftarrow \text{MULTI COLLISION}(h_1, a_0)$ 
  for  $0 \leq k < T$  do
     $b_k \leftarrow h_2^*(b_0, \mathcal{M}_0[k])$ 
  end for
   $a_0 \leftarrow h_1^*(a_0, \mathcal{M}_0[0])$ 
   $\mathcal{M}_1 \leftarrow \text{MULTI COLLISION}(h_2, b_0)$ 
  for  $1 \leq k < T$  do
     $a_k \leftarrow h_2^*(a_0, \mathcal{M}_1[k])$ 
  end for
   $a_0 \leftarrow h_2^*(a_0, \mathcal{M}_1[0])$ 
   $b_0 \leftarrow h_1^*(b_0, \mathcal{M}_1[0])$ 
  for  $2 \leq j < T$  do
    for  $1 \leq i < T$  do
       $(M, M') \leftarrow \text{SWITCH}(h_2, h_1, b_j, a_0, a_i)$ 
       $M \leftarrow M \parallel M; M' \leftarrow M' \parallel M'$ 
      for  $0 \leq k < T$  do
         $a_k \leftarrow h_1^*(a_k, M)$ 
         $b_k \leftarrow h_2^*(b_k, M)$ 
      end for
    end for
  end for
  return  $(\mathcal{M}_0, \mathcal{M}_1, M, M')$ 
end function

function SELECTMESSAGE( $\mathcal{M}_0, \mathcal{M}_1, M, M', j, k$ )
  if  $j = 0$  then
    return  $\mathcal{M}_0[0] \parallel \mathcal{M}_1[0] \parallel M$ 
  else if  $k = 0$  then
    return  $\mathcal{M}_0[0] \parallel \mathcal{M}_1[j] \parallel M$ 
  else
     $\mu \leftarrow M$ 
     $\mu[(k-1) \cdot (T-1) + j - 1] \leftarrow M'[(k-1) \cdot (T-1) + j - 1]$ 
    return  $\mathcal{M}_0[0] \parallel \mathcal{M}_1[0] \parallel \mu$ 
  end if
end function

function MULTI COLLISION( $h, x$ )
   $\mathcal{M} \leftarrow \{\}$ 
  for  $0 \leq i < n/2$  do
     $(m, m') \leftarrow \text{COLLISION}(h, x)$ 
     $x \leftarrow h(x, m)$ 
     $\mathcal{M} \leftarrow (\mathcal{M} \parallel m) \cup (\mathcal{M} \parallel m')$ 
  end for
  return  $\mathcal{M}$ 
end function

```

References

1. Andreeva, E., Bouillaguet, C., Dunkelman, O., Kelsey, J.: Herding, Second Preimage and Trojan Message Attacks Beyond Merkle-Damgård. In: Jacobson Jr, M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 393–414. Springer, Heidelberg (2009)
2. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions - HAIFA. IACR Cryptology ePrint Archive, Report 2007/278 (2007)
3. Boneh, D., Boyen, X.: On the Impossibility of Efficiently Combining Collision Resistant Hash Functions. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 570–583. Springer, Heidelberg (2006)
4. Damgård, I.B.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
5. Dierks, T., Allen, C.: The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, January 1999. <http://www.ietf.org/rfc/rfc2246.txt>
6. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), updated by RFCs 5746, 5878, 6176, August 2008. <http://www.ietf.org/rfc/rfc5246.txt>
7. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A Strengthened Version of RIPEMD. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 71–82. Springer, Heidelberg (1996)
8. Fischlin, M., Lehmann, A.: Multi-Property Preserving Combiners for Hash Functions. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 375–392. Springer, Heidelberg (2008)
9. Fischlin, M., Lehmann, A., Pietrzak, K.: Robust Multi-Property Combiners for Hash Functions Revisited. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórssón, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 655–666. Springer, Heidelberg (2008)
10. Fischlin, M., Lehmann, A., Pietrzak, K.: Robust Multi-Property Combiners for Hash Functions. *J. Cryptology* **27**(3), 397–428 (2014)
11. Freier, A., Karlton, P., Kocher, P.: The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011. <http://www.ietf.org/rfc/rfc6101.txt>
12. Her, Y.S., Sakurai, K.: A Design of Cryptographic Hash Function Group with Variable Output-Length Based on SHA-1. Technical report of IEICE. ISEC 102(212), pp. 69–76, July 2002. <http://ci.nii.ac.jp/naid/110003298501/en/>
13. Hoch, J.J., Shamir, A.: Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 179–194. Springer, Heidelberg (2006)
14. Hoch, J.J., Shamir, A.: On the Strength of the Concatenated Hash Combiner When All the Hash Functions are Weak. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórssón, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 616–630. Springer, Heidelberg (2008)
15. Hong, D., Chang, D., Sung, J., Lee, S.-J., Hong, S.H., Lee, J.S., Moon, D., Chee, S.: A New Dedicated 256-bit Hash Function: FORK-256. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 195–209. Springer, Heidelberg (2006)
16. Indesteege, S.: The lane hash function. Submission to NIST (2008). <http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf>
17. Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)

18. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In: Vaudenay, S. (ed.) *EUROCRYPT 2006*. LNCS, vol. 4004, pp. 183–200. Springer, Heidelberg (2006)
19. Kelsey, J., Schneier, B.: Second Preimages on n -bit Hash Functions for Much Less than 2^n Work. In: Cramer, R. (ed.) *EUROCRYPT 2005*. LNCS, vol. 3494, pp. 474–490. Springer, Heidelberg (2005)
20. Lehmann, A.: On the Security of Hash Function Combiners. Ph.D. thesis, TU Darmstadt (2010)
21. Liskov, M.: Constructing An Ideal Hash Function from Weak Ideal Compression Functions. In: Biham, E., Youssef, A.M. (eds.) *SAC 2006*. LNCS, vol. 4356, pp. 358–375. Springer, Heidelberg (2007)
22. Mendel, F., Nad, T., Scherz, S., Schläffer, M.: Differential Attacks on Reduced RIPEMD-160. In: Gollmann, D., Freiling, F.C. (eds.) *ISC 2012*. LNCS, vol. 7483, pp. 23–38. Springer, Heidelberg (2012)
23. Mendel, F., Peyrin, T., Schläffer, M., Wang, L., Wu, S.: Improved Cryptanalysis of Reduced RIPEMD-160. In: Sako, K., Sarkar, P. (eds.) *ASIACRYPT 2013*, Part II. LNCS, vol. 8270, pp. 484–503. Springer, Heidelberg (2013)
24. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: On the Collision Resistance of RIPEMD-160. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) *ISC 2006*. LNCS, vol. 4176, pp. 101–116. Springer, Heidelberg (2006)
25. Mendel, F., Rechberger, C., Schläffer, M.: MD5 is Weaker than Weak: Attacks on Concatenated Combiners. In: Matsui, M. (ed.) *ASIACRYPT 2009*. LNCS, vol. 5912, pp. 144–161. Springer, Heidelberg (2009)
26. Mennink, B., Preneel, B.: Breaking and Fixing Cryptophia’s Short Combiner. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) *CANS 2014*. LNCS, vol. 8813, pp. 50–63. Springer, Heidelberg (2014)
27. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) *CRYPTO 1989*. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg (1990)
28. Mittelbach, A.: Hash Combiners for Second Pre-Image Resistance, Target Collision Resistance and Pre-Image Resistance have Long Output. In: Visconti, I., De Prisco, R. (eds.) *SCN 2012*. LNCS, vol. 7485, pp. 522–539. Springer, Heidelberg (2012)
29. Mittelbach, A.: Cryptophia’s Short Combiner for Collision-Resistant Hash Functions. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) *ACNS 2013*. LNCS, vol. 7954, pp. 136–153. Springer, Heidelberg (2013)
30. Nandi, M., Stinson, D.R.: Multicollision Attacks on Some Generalized Sequential Hash Functions. *IEEE Transactions on Information Theory* **53**(2), 759–767 (2007)
31. Park, N.K., Hwang, J.H., Lee, P.J.: HAS-V: A New Hash Function with Variable Output Length. In: Stinson, D.R., Tavares, S. (eds.) *SAC 2000*. LNCS, vol. 2012, pp. 202–216. Springer, Heidelberg (2001)
32. Pietrzak, K.: Non-Trivial Black-Box Combiners for Collision-Resistant Hash-Functions Don’t Exist. In: Naor, M. (ed.) *EUROCRYPT 2007*. LNCS, vol. 4515, pp. 23–33. Springer, Heidelberg (2007)
33. Rjasko, M.: On existence of robust combiners for cryptographic hash functions. In: Vojtás, P. (ed.) *ITAT. CEUR Workshop Proceedings*, vol. 584, pp. 71–76. CEUR-WS.org (2009)
34. Sasaki, Y., Wang, L.: Distinguishers Beyond Three Rounds of the Ripemd-128-/160 Compression Functions. In: Bao, F., Samarati, P., Zhou, J. (eds.) *ACNS 2012*. LNCS, vol. 7341, pp. 275–292. Springer, Heidelberg (2012)

35. Wagner, D.: A Generalized Birthday Problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–304. Springer, Heidelberg (2002)
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)
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)

SPHINCS: Practical Stateless Hash-Based Signatures

Daniel J. Bernstein^{1,3(✉)}, Daira Hopwood^{2(✉)}, Andreas Hülsing^{3(✉)},
Tanja Lange^{3(✉)}, Ruben Niederhagen^{3(✉)}, Louiza Papachristodoulou^{4(✉)},
Michael Schneider, Peter Schwabe^{4(✉)}, and Zooko Wilcox-O’Hearn^{2(✉)}

¹ Department of Computer Science, University of Illinois at Chicago, Chicago, IL
60607–7045, USA

djb@cr.yp.to

² Least Authority, 3450 Emerson Ave., Boulder, CO 80305–6452, USA
{daira,zooko}@leastauthority.com

³ Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB
Eindhoven, The Netherlands
tanja@hyperelliptic.org, ruben@polycephaly.org,
andreas.huelsing@googlemail.com

⁴ Digital Security Group, Radboud University Nijmegen, P.O. Box 9010,
6500 GL Nijmegen, The Netherlands
louiza@cryptologic.org, peter@cryptojedi.org

Abstract. This paper introduces a high-security post-quantum stateless hash-based signature scheme that signs hundreds of messages per second on a modern 4-core 3.5GHz Intel CPU. Signatures are 41 KB, public keys are 1 KB, and private keys are 1 KB. The signature scheme is designed to provide long-term 2^{128} security even against attackers equipped with quantum computers. Unlike most hash-based designs, this signature scheme is stateless, allowing it to be a drop-in replacement for current signature schemes.

Keywords: Post-quantum cryptography · One-time signatures · Few-time signatures · Hypertrees · Vectorized implementation

1 Introduction

It is not at all clear how to securely sign operating-system updates, web-site certificates, etc. once an attacker has constructed a large quantum computer:

- RSA and ECC are perceived today as being small and fast, but they are broken in polynomial time by Shor’s algorithm. The polynomial is so small that scaling up to secure parameters seems impossible.
- Lattice-based signature schemes are reasonably fast and provide reasonably small signatures and keys for proposed parameters. However, their quantitative security levels are highly unclear. It is unsurprising for a lattice-based scheme to promise “100-bit” security for a parameter set in 2012 and to

correct this promise to only “75–80 bits” in 2013 (see [19, footnote 2]). Furthermore, both of these promises are only against pre-quantum attacks, and it seems likely that the same parameters will be breakable in practice by quantum computers.

- Multivariate-quadratic signature schemes have extremely short signatures, are reasonably fast, and in some cases have public keys short enough for typical applications. However, the long-term security of these schemes is even less clear than the security of lattice-based schemes.
- Code-based signature schemes provide short signatures, and in some cases have been studied enough to support quantitative security conjectures. However, the schemes that have attracted the most security analysis have keys of many megabytes, and would need even larger keys to be secure against quantum computers.

Hash-based signature schemes are perhaps the most attractive answer. Every signature scheme uses a cryptographic hash function; hash-based signatures use nothing else. Many hash-based signature schemes offer security proofs relative to comprehensible, and plausible, properties of the hash function, properties that have not been broken even when the hash function is MD5. (We do not mean to suggest that MD5 is a good choice of hash function; it is easy to make, and we recommend, much more conservative parameter choices.) A recent result by Song [35] shows that these proofs are still valid for quantum adversaries; this is not known to be the case for many other post-quantum signature proposals. Hash-based signing is reasonably fast, even without hardware acceleration; verification is faster; signatures and keys are reasonably small.

However, every practical hash-based signature scheme in the literature is *stateful*. Signing reads a secret key and a message and generates a signature but also generates an updated secret key. This does not fit standard APIs; it does not even fit the standard *definition* of signatures in cryptography. If the update fails (for example, if a key is copied from one device to another, or backed up and later restored) then security disintegrates.

It has been known for many years that, as a theoretical matter, one can build hash-based signature schemes without a state. What we show in this paper is that high-security post-quantum stateless hash-based signature systems are *practical*, and in particular that they can sign hundreds of messages per second on a modern 4-core 3.5GHz Intel CPU using parameters that provide 2^{128} security against quantum attacks. In particular, we

- introduce SPHINCS, a new method to do randomized tree-based stateless signatures;
- introduce HORS with trees (HORST), an improvement of the HORS few-time signature scheme;
- propose SPHINCS-256, an efficient high-security instantiation of SPHINCS; and
- describe a fast vectorized implementation of SPHINCS-256.

SPHINCS is carefully designed so that its security can be based on weak standard-model assumptions, avoiding collision resistance and the random-oracle model.

Hash-Based Signatures. The idea of hash-based signatures goes back to a proposal from 1979 by Lamport [30]. In Lamport’s scheme, the public key consists of two hash outputs for secret inputs; to sign bit 0, reveal the preimage of the first output; to sign bit 1, reveal the preimage of the second output. Obviously the secret key in this scheme can be used only once: signing a total of T bits of messages requires a sequence of T public keys and is therefore highly impractical.

To allow a short public key to sign many messages, Merkle [32] proposed what are now called Merkle trees. Merkle starts with a one-time signature scheme (OTS), i.e. a signature scheme where a key pair is used only once. To construct a many-time signature scheme, Merkle authenticates 2^h OTS key pairs using a binary hash tree of height h . The leaves of this tree are the hashes of the OTS public keys. The OTS secret keys become the secret key of the new scheme and the root of the tree the public key. A key pair can be used to sign 2^h messages.

A signature of the many-time signature scheme is also called a *full* signature if necessary to distinguish it from other kinds of signatures. A full signature contains the index of the used OTS key pair in the tree; the OTS public key; the OTS signature; and the *authentication path*, i.e., the set of sibling nodes on the path from the OTS public key to the root. (If a Winternitz-style OTS is used, the OTS public key can be computed from the OTS signature. Hence, the OTS public key can be omitted in the full signature in that case.) To guarantee that each OTS key pair is used only once, the OTS key pairs are used in a predefined order, using the leaves of the tree from left to right. To verify the signature, one verifies the OTS signature on the message, and verifies the authenticity of the OTS key pair by checking whether the public key is consistent with the authentication path and the hash of the OTS public key.

This approach generates small signatures, small secret keys (using pseudo-random generation of the OTS secret keys), and small public keys. However, key generation and signature time are exponential in h as the whole tree has to be built in the key generation. Recent practical hash-based signature systems [15–18, 27] solve these two performance problems. First, key generation time is significantly reduced using a hyper-tree of several layers of trees, i.e. a certification tree where a single hash tree of height h_1 is used to sign the public keys of 2^{h_1} hash-based key pairs and so on. During key generation only one tree on each layer has to be generated. Using d layers of trees with height h/d reduces the key-generation time from $\mathcal{O}(2^h)$ to $\mathcal{O}(d2^{h/d})$. Second, signing time is reduced from $\mathcal{O}(2^h)$ to $\mathcal{O}(h)$ using stateful algorithms that exploit the ordered use of the OTS key pairs. When combined with hyper-trees, the ordered use of the trees reduces the signing time even further to $\mathcal{O}(h/d)$.

From Stateful to Stateless. Goldreich [23] (elaborating upon [22]) proposed a *stateless* hash-based signature scheme, using a binary certification tree built out of one-time signature keys. In Goldreich’s system, each OTS key pair

corresponding to a non-leaf node is used to sign the hash of the public keys of its two child nodes. The leaf OTS key pairs are used to sign the messages. The OTS public key of the root becomes the overall public key. The secret key is a seed value that is used to pseudorandomly generate all the OTS key pairs of the tree.

It is important to ensure that a single OTS key pair is never used to sign two different messages. One approach is to sign message M using a tree of height n as follows: compute an n -bit hash of M , view the hash as an integer h between 0 and $2^n - 1$, and use the leaf at index h to sign. The full signature contains all the OTS public keys in the path from leaf h to the root, all the public keys of the sibling nodes on this path, and the one-time signatures on the message and on the public keys in the path. Security obviously cannot be better than the collision resistance of the hash function, at most $2^{n/2}$.

For this scheme, key generation requires a single OTS key generation. Signing takes $2n$ OTS key generations and n OTS signatures. This can be done in reasonable time for secure parameters. Keys are also very short: one OTS public key ($\mathcal{O}(n^2)$) for the public key and a single seed value ($\mathcal{O}(n)$) for the secret key. However, the signature size is cubic in the security parameter. Consider, for example, the Winternitz OTS construction from [26], which has small signatures for a hash-based OTS; taking $n = 256$ as we do for SPHINCS-256, and applying some straightforward optimizations, produces a Goldreich signature size that is still above 1 MB.

One way to evaluate the real-world impact of particular signature sizes is to compare those sizes to the sizes of messages being signed. For example, in the Debian operating system (September 2014 Wheezy distribution), the average package size is 1.2 MB and the median package size is just 0.08 MB. Debian is designed for frequent updates, typically upgrading just one package or a few packages at a time, and of course each upgrade has to check at least one new signature. As another example, the size of an average web page in the Alexa Top 1000000 is 1.8 MB, and HTTPS typically sends multiple signatures per page; the exact number depends on how many HTTPS sites cooperate to produce the page, how many certificates are sent, etc. A signature size above 1 MB would often dominate the traffic in these applications and would also add user-visible latency on typical network connections.

Goldreich also proposes randomized leaf selection: instead of applying a public hash function to the message to determine the index of the OTS key pair, select an index randomly. It is then safe for the total tree height h to be somewhat smaller than the hash output length n : the hash output length protects against offline computations by the attacker, while the tree height protects against accidental collisions in indices chosen by the signer. For example, choosing h as 128 instead of 256 saves a factor of 2 in signature size and signing speed, if it is acceptable to have probability roughly 2^{-30} of OTS reuse (presumably breaking the system) within 2^{50} signatures.

The SPHINCS Approach. SPHINCS introduces two new ideas that together drastically reduce signature size. First, to increase the security level of randomized

index selection, SPHINCS replaces the leaf OTS with a hash-based *few-time* signature scheme (FTS). An FTS is, as the name suggests, a signature scheme designed to sign a few messages; in the context of SPHINCS this allows a few index collisions, which in turn allows a smaller tree height for the same security level. For our FTS (see below) the probability of a forgery after γ signatures gradually increases with γ , while the probability that the signer uses the same FTS key γ times gradually decreases with γ ; we choose parameters to make sure that the product of these probabilities is sufficiently small for all $\gamma \in \mathbb{N}$. For example, SPHINCS-256 reduces the total tree height from 256 to just 60 while maintaining 2^{128} security against quantum attackers.

Second, SPHINCS views Goldreich's construction as a hyper-tree construction with h layers of trees of height 1, and generalizes to a hyper-tree with d layers of trees of height h/d . This introduces a tradeoff between signature size and time controlled by the number of layers d . The signature size is $|\sigma| \approx d|\sigma_{\text{OTS}}| + hn$ assuming a hash function with n -bit outputs. Recall that the size of a one-time signature $|\sigma_{\text{OTS}}|$ is roughly $\mathcal{O}(n^2)$, so by decreasing the number of layers we get smaller full signatures. The tradeoff is that signing time increases exponentially in the decrease of layers: signing takes $d2^{h/d}$ OTS key generations and $d2^{h/d} - d$ hash computations. For example, in SPHINCS-256, with $h = 60$, we reduce d from 60 to 12, increasing $d2^{h/d}$ from 120 to 384.

We accompany our construction with an exact security reduction to some standard-model properties of hash functions. For parameter selection, we analyze the costs of generic attacks against these properties when the attacker has access to a large-scale quantum computer. For SPHINCS-256 we select parameters that provide 128 bits of security against quantum attackers and keep a balance between signature size and time.

HORS and HORST. HORS [34] is a fast hash-based FTS. For message hashes of length m , HORS uses two parameters $t = 2^\tau$ for $\tau \in \mathbb{N}$ and $k \in \mathbb{N}$ such that $m = k \log t = k\tau$. For practical secure parameters $t \gg k$. HORS uses a secret key consisting of t random values. The public key consists of the t hashes of these values. A signature consists of k secret key elements, with indices selected as public functions of the message being signed.

In the context of SPHINCS, each full signature has to include not just an FTS signature but also an FTS public key. The problem with HORS is that it has large public keys. Of course, one can replace the public key in any signature system by a short hash of the original public key, but then the original public key needs to be included in the signature; this does not improve the total length of key and signature.

As a better FTS for SPHINCS we introduce HORS with trees (HORST). Compared to HORS, HORST sacrifices runtime to reduce the public key size and the combined size of a signature and a public key. A HORST public key is the root node of a binary hash tree of height $\log t$, where the leaves are the public key elements of a HORS key. This reduces the public key size to a single hash value. For this to work, a HORST signature contains not only the k secret key elements but also one authentication path per secret key element. Now the public

key can be computed given a signature. A full hash-based signature thus includes just $k \log t$ hash values for HORST, compared to t hash values for HORS. We also introduce some optimizations that further compress signatures.

For the SPHINCS-256 parameters, switching from HORS to HORST reduces the FTS part of the full signature from 2^{16} hash values to fewer than $16 \cdot 32 = 2^9$ hash values, i.e., from 2 MB to just 16 KB.

The same idea applies in more generality. For example, HORS++ [33] is a variant of HORS that gives stronger security guarantees but that has bigger keys; the changes from HORS to HORST can easily be adapted to HORS++, producing HORST++.

Vectorized Software Implementation. We also present an optimized implementation of SPHINCS-256. Almost all hash computations in SPHINCS are highly parallel and we make extensive use of vector instructions. On an Intel Xeon E3-1275 (Haswell) CPU, our hashing throughput is about 1.6 cycles/byte. Signing a short message with SPHINCS-256 takes 51 636 372 cycles on a single core; simultaneous signing on all 4 cores of the 3.5 GHz CPU has a throughput of more than 200 signatures per second, fast enough for most applications. Verification takes only 1 451 004 cycles; key-pair generation takes 3 237 260 cycles.

We placed the software described in this paper into the public domain to maximize reusability of our results. We submitted the software to eBACS [10] for independent benchmarking; the software is also available online at <http://cryptojedi.org/crypto/#sphincs>.

Notation. We always use the logarithm with base 2 and hence write \log instead of \log_2 . We write $[x]$ for the set $\{0, 1, \dots, x\}$. Given a bit string x we write $x(i)$ for the i th bit of x and $x(i, j)$ for the j -bit substring of x that starts with the i th bit.

2 The SPHINCS Construction

In this section we describe our main construction. We begin by listing the parameters used in the construction, reviewing the one-time signature scheme WOTS⁺, and reviewing binary hash trees. In Section 2.1 we present our few-time signature scheme HORST, and in Section 2.2 we present our many-time signature scheme SPHINCS.

Parameters. SPHINCS uses several parameters and several functions. The main security parameter is $n \in \mathbb{N}$. The functions include two short-input cryptographic hash functions $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$ and $H : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$; one arbitrary-input randomized hash function $\mathcal{H} : \{0, 1\}^n \times \{0, 1\}^* \rightarrow \{0, 1\}^m$, for $m = \text{poly}(n)$; a family of pseudorandom generators $G_\lambda : \{0, 1\}^n \rightarrow \{0, 1\}^{\lambda n}$ for different values of λ ; an ensemble of pseudorandom function families $\mathcal{F}_\lambda : \{0, 1\}^\lambda \times \{0, 1\}^n \rightarrow \{0, 1\}^n$; and a pseudorandom function family $\mathcal{F} : \{0, 1\}^* \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ that supports arbitrary input lengths. Of course, these functions can all be built from a single cryptographic hash function, but it is more natural to separate the functions according to their roles.

SPHINCS uses a hyper-tree (a tree of trees) of total height $h \in \mathbb{N}$, where h is a multiple of d and the hyper-tree consists of d layers of trees, each having height h/d . The components of SPHINCS have additional parameters which influence performance and size of the signature and keys: the Winternitz one-time signature WOTS naturally allows for a space-time tradeoff using the Winternitz parameter $w \in \mathbb{N}, w > 1$; the tree-based few-time signature scheme HORST has a space-time tradeoff which is controlled by two parameters $k \in \mathbb{N}$ and $t = 2^\tau$ with $\tau \in \mathbb{N}$ and $k\tau = m$.

As a running example we present concrete numbers for SPHINCS-256; the choices are explained in Section 4. For SPHINCS-256 we use $n = 256, m = 512, h = 60, d = 12, w = 16, t = 2^{16}, k = 32$.

WOTS⁺. We now describe the Winternitz one-time signature (WOTS⁺) from [26]. We deviate slightly from the description in [26] to describe the algorithms as they are used in SPHINCS. Specifically, we include pseudorandom key generation and fix the message length to be n , meaning that a seed value takes the place of a secret key in our description. Given n and w , we define

$$\ell_1 = \left\lceil \frac{n}{\log(w)} \right\rceil, \quad \ell_2 = \left\lfloor \frac{\log(\ell_1(w-1))}{\log(w)} \right\rfloor + 1, \quad \ell = \ell_1 + \ell_2.$$

For the SPHINCS-256 parameters this leads to $\ell = 67$. WOTS⁺ uses the function F to construct the following chaining function.

Chaining function $c^i(x, \mathbf{r})$: On input of value $x \in \{0, 1\}^n$, iteration counter $i \in \mathbb{N}$, and bitmasks $\mathbf{r} = (r_1, \dots, r_j) \in \{0, 1\}^{n \times j}$ with $j \geq i$, the chaining function works the following way. In case $i = 0$, c returns x , i.e., $c^0(x, \mathbf{r}) = x$. For $i > 0$ we define c recursively as

$$c^i(x, \mathbf{r}) = F(c^{i-1}(x, \mathbf{r}) \oplus r_i),$$

i.e. in every round, the function first takes the bitwise xor of the previous value $c^{i-1}(x, \mathbf{r})$ and bitmask r_i and evaluates F on the result. We write $\mathbf{r}_{a,b}$ for the substring (r_a, \dots, r_b) of \mathbf{r} . In case $b < a$ we define $\mathbf{r}_{a,b}$ to be the empty string. Now we describe the three algorithms of WOTS⁺.

Key Generation Algorithm ($\mathbf{sk}, \mathbf{pk} \leftarrow \text{WOTS}.\text{kg}(\mathcal{S}, \mathbf{r})$): On input of seed $\mathcal{S} \in \{0, 1\}^n$ and bitmasks $\mathbf{r} \in \{0, 1\}^{n \times (w-1)}$ the key generation algorithm computes the internal secret key as $\mathbf{sk} = (\mathbf{sk}_1, \dots, \mathbf{sk}_\ell) \leftarrow G_\ell(\mathcal{S})$, i.e., the n bit seed is expanded to ℓ values of n bits. The public key \mathbf{pk} is computed as

$$\mathbf{pk} = (\mathbf{pk}_1, \dots, \mathbf{pk}_\ell) = (c^{w-1}(\mathbf{sk}_1, \mathbf{r}), \dots, c^{w-1}(\mathbf{sk}_\ell, \mathbf{r})).$$

Note that \mathcal{S} requires less storage than \mathbf{sk} ; thus we generate \mathbf{sk} and \mathbf{pk} on the fly when necessary.

Signature Algorithm ($\sigma \leftarrow \text{WOTS}.\text{sign}(M, \mathcal{S}, \mathbf{r})$): On input of an n -bit message M , seed \mathcal{S} and the bitmasks \mathbf{r} , the signature algorithm first computes a base- w representation of M : $M = (M_1 \dots M_{\ell_1})$, $M_i \in \{0, \dots, w-1\}$. That is, M

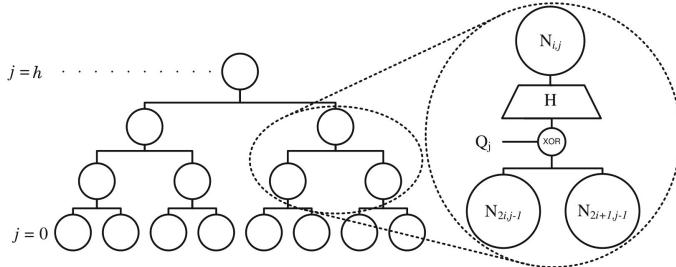


Fig. 1. The binary hash tree construction

is treated as the binary representation of a natural number x and then the w -ary representation of x is computed. Next it computes the checksum $C = \sum_{i=1}^{\ell_1} (w - 1 - M_i)$ and its base w representation $C = (C_1, \dots, C_{\ell_2})$. The length of the base w representation of C is at most ℓ_2 since $C \leq \ell_1(w - 1)$. We set $B = (b_1, \dots, b_\ell) = M \parallel C$, the concatenation of the base w representations of M and C . Then the internal secret key is generated using $G_\ell(\mathcal{S})$ the same way as during key generation. The signature is computed as

$$\sigma = (\sigma_1, \dots, \sigma_\ell) = (c^{b_1}(\mathbf{sk}_1, \mathbf{r}), \dots, c^{b_\ell}(\mathbf{sk}_\ell, \mathbf{r})).$$

Verification Algorithm ($\mathbf{pk}' \leftarrow \text{WOTS.vf}(M, \sigma, \mathbf{r})$): On input of an n -bit message M , a signature σ , and bitmasks \mathbf{r} , the verification algorithm first computes the b_i , $1 \leq i \leq \ell$ as described above. Then it returns:

$$\mathbf{pk}' = (\mathbf{pk}'_1, \dots, \mathbf{pk}'_\ell) = (c^{w-1-b_1}(\sigma_1, \mathbf{r}_{b_1+1, w-1}), \dots, c^{w-1-b_\ell}(\sigma_\ell, \mathbf{r}_{b_\ell+1, w-1})).$$

A formally correct verification algorithm would compare \mathbf{pk}' to a given public key and output **true** on equality and **false** otherwise. In SPHINCS this comparison is delegated to the overall verification algorithm.

Binary Hash Trees. The central elements of our construction are full binary hash trees. We use the construction proposed in [18] shown in Figure 1.

In SPHINCS, a binary hash tree of height h always has 2^h leaves which are n bit strings L_i , $i \in [2^h - 1]$. Each node $N_{i,j}$, for $0 < j \leq h$, $0 \leq i < 2^{h-j}$, of the tree stores an n -bit string. To construct the tree, h bit masks $\mathbf{Q}_j \in \{0, 1\}^{2n}$, $0 < j \leq h$, are used. For the leaf nodes define $N_{i,0} = L_i$. The values of the internal nodes $N_{i,j}$ are computed as

$$N_{i,j} = H((N_{2i,j-1} || N_{2i+1,j-1}) \oplus \mathbf{Q}_j).$$

We also denote the root as $\text{ROOT} = N_{0,h}$.

An important notion is the authentication path $\mathbf{Auth}_i = (\mathbf{A}_0, \dots, \mathbf{A}_{h-1})$ of a leaf L_i shown in Figure 2. \mathbf{Auth}_i consists of all the sibling nodes of the nodes contained in the path from L_i to the root. For a discussion on how to compute

Input: Leaf index i , leaf L_i , authentication path $\text{Auth}_i = (\mathbf{A}_0, \dots, \mathbf{A}_{h-1})$ for L_i .
Output: Root node ROOT of the tree that contains L_i .

```

Set  $P_0 \leftarrow L_i$ ;
for  $j \leftarrow 1$  up to  $h$  do
  
$$P_j = \begin{cases} H((P_{j-1} || \mathbf{A}_{j-1}) \oplus \mathbf{Q}_j), & \text{if } \lfloor i/2^{j-1} \rfloor \equiv 0 \pmod{2}; \\ H((\mathbf{A}_{j-1} || P_{j-1}) \oplus \mathbf{Q}_j), & \text{if } \lfloor i/2^{j-1} \rfloor \equiv 1 \pmod{2}; \end{cases}$$

end
return  $P_h$ 
```

Algorithm 1. Root Computation

authentication paths, see Section 5. Given a leaf L_i together with its authentication path Auth_i , the root of the tree can be computed using Algorithm 1.

L-Tree. In addition to the full binary trees above, we also use unbalanced binary trees called L-Trees as in [18]. These are exclusively used to hash WOTS⁺ public keys. The ℓ leaves of an L-Tree are the elements of a WOTS⁺ public key and the tree is constructed as described above but with one difference: A left node that has no right sibling is lifted to a higher level of the L-Tree until it becomes the right sibling of another node. Apart from this the computations work the same as for binary trees. The L-Trees have height $\lceil \log \ell \rceil$ and hence need $\lceil \log \ell \rceil$ bitmasks.

2.1 HORST

HORST signs messages of length m and uses parameters k and $t = 2^\tau$ with $k\tau = m$ (typical values as used in SPHINCS-256 are $t = 2^{16}, k = 32$). HORST improves HORS [34] using a binary hash-tree to reduce the public key size from tn bits to n bits¹ and the combined signature and public key size from tn bits to $(k(\tau - x + 1) + 2^x)n$ bits for some $x \in \mathbb{N} \setminus \{0\}$. The value x is determined based on t and k such that $k(\tau - x + 1) + 2^x$ is minimal. It might happen that the expression takes its minimum for two successive values. In this case the greater value is used. For SPHINCS-256 this results in $x = 6$.

In contrast to a one-time signature scheme like WOTS, HORST can be used to sign more than one message with the same key pair. However, with each signature the security decreases. See Section 3 for more details. Like for WOTS⁺ our description includes pseudorandom key generation. We now describe the algorithms for HORST:

Key Generation Algorithm ($\text{pk} \leftarrow \text{HORST.kg}(\mathcal{S}, \mathbf{Q})$): On input of seed $\mathcal{S} \in \{0, 1\}^n$ and bitmasks $\mathbf{Q} \in \{0, 1\}^{2n \times \log t}$ the key generation algorithm first computes the internal secret key $\text{sk} = (\text{sk}_1, \dots, \text{sk}_t) \leftarrow \text{G}_t(\mathcal{S})$. The leaves of the tree are computed as $L_i = F(\text{sk}_i)$ for $i \in [t - 1]$ and the tree is constructed using

¹ Here we assume that the used bitmasks are given as they are used for several key pairs. Otherwise, public key size is $(2\tau + 1)n$ bit including bitmasks, which is still less than tn bits.

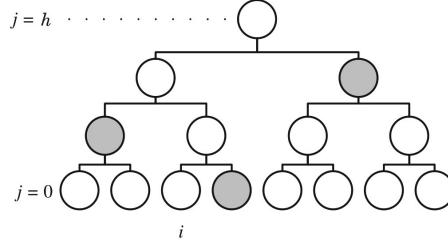


Fig. 2. The authentication path for leaf L_i

bitmasks \mathbf{Q} . The public key \mathbf{pk} is computed as the root node of a binary tree of height $\log t$.

Signature Algorithm ($(\sigma, \mathbf{pk}) \leftarrow \text{HORST}.\text{sign}(M, \mathcal{S}, \mathbf{Q})$): On input of a message $M \in \{0, 1\}^m$, seed $\mathcal{S} \in \{0, 1\}^n$, and bitmasks $\mathbf{Q} \in \{0, 1\}^{2n \times \log t}$ first the internal secret key \mathbf{sk} is computed as described above. Then, let $M = (M_0, \dots, M_{k-1})$ denote the k numbers obtained by splitting M into k strings of length $\log t$ bits each and interpreting each as an unsigned integer. The signature $\sigma = (\sigma_0, \dots, \sigma_{k-1}, \sigma_k)$ consists of k blocks $\sigma_i = (\mathbf{sk}_{M_i}, \mathbf{Auth}_{M_i})$ for $i \in [k-1]$ containing the M_i th secret key element and the lower $\tau - x$ elements of the authentication path of the corresponding leaf $(\mathbf{A}_0, \dots, \mathbf{A}_{\tau-1-x})$. The block σ_k contains all the 2^x nodes of the binary tree on level $\tau - x$ ($N_{0,\tau-x}, \dots, N_{2^x-1,\tau-x}$). In addition to the signature, HORST.sign also outputs the public key.

Verification Algorithm ($\mathbf{pk}' \leftarrow \text{HORST}.\text{vf}(M, \sigma, \mathbf{Q})$): On input of message $M \in \{0, 1\}^m$, a signature σ , and bitmasks $\mathbf{Q} \in \{0, 1\}^{2n \times \log t}$, the verification algorithm first computes the M_i , as described above. Then, for $i \in [k-1]$, $y_i = \lfloor M_i / 2^\tau - x \rfloor$ it computes $N'_{y_i, \tau-x}$ using Algorithm 1 with index M_i , $L_{M_i} = F(\sigma_i^1)$, and $\mathbf{Auth}_{M_i} = \sigma_i^2$. It then checks that $\forall i \in [k-1] : N'_{y_i, \tau-x} = N_{y_i, \tau-x}$, i.e., that the computed nodes match those in σ_k . If all comparisons hold it uses σ_k to compute and then return ROOT_0 , otherwise it returns fail.

Theoretical Performance. In the following we give rough theoretical performance values for HORST when used in a many-time signature scheme. We ignore the space needed for bitmasks, assuming they are provided. For runtimes we only count PRG calls and the number of hash evaluations without distinguishing the different hash functions.

Sizes: A HORST secret key consists of a single n bit seed. The public key contains a single n bit hash. A signature contains k secret key elements and authentication paths of length $(\log t) - x$ (Recall $t = 2^\tau$ is a power of two). In addition it contains 2^x nodes in σ_k , adding up to a total of $(k((\log t) - x + 1) + 2^x)n$ bits.

Runtimes: Key generation needs one evaluation of G_t and t hashes to compute the leaf values and $t - 1$ hashes to compute the public key, leading to a total of $2t - 1$. Signing takes the same time as we require the root is part of the output. Verification takes k times one hash to compute a leaf value plus $(\log t) - x$ hashes to compute the node on level $(\log t) - x$. In addition, $2^x - 1$ hashes are needed to compute the root from σ_k . Together these are $k((\log t) - x + 1) + 2^x - 1$ hashes.

2.2 SPHINCS

Given all of the above we can finally describe the algorithms of the SPHINCS construction. A SPHINCS keypair completely defines a “virtual” structure which we explain first. SPHINCS works on a hyper-tree of height h that consists of d layers of trees of height h/d . Each of these trees looks as follows. The leaves of a tree are $2^{h/d}$ L-Tree root nodes that each compress the public key of a WOTS⁺ key pair. Hence, a tree can be viewed as a key pair that can be used to sign $2^{h/d}$ messages. The hyper-tree is structured into d layers. On layer $d - 1$ it has a single tree. On layer $d - 2$ it has $2^{h/d}$ trees. The roots of these trees are signed using the WOTS⁺ key pairs of the tree on layer $d - 1$. In general, layer i consists of $2^{(d-1-i)(h/d)}$ trees and the roots of these trees are signed using the WOTS⁺ key pairs of the trees on layer $i + 1$. Finally, on layer 0 each WOTS⁺ key pair is used to sign a HORST public key. We talk about a “virtual” structure as all values within are determined choosing a seed and the bitmasks, and as the full structure is never computed. The seed is part of the secret key and used for pseudorandom key generation. To support easier understanding, Figure 3 shows the virtual structure of a SPHINCS signature, i.e. of one path inside the hyper-tree. It contains d trees TREE_i , $i \in [d - 1]$ (each consisting of a binary hash tree that authenticates the root nodes of $2^{h/d}$ L-Trees which in turn each have the public key nodes of one WOTS⁺ keypair as leaves). Each tree authenticates the tree below using a WOTS⁺ signature $\sigma_{W,i}$. The only exception is TREE_0 which authenticates a HORST public key using a WOTS⁺ signature. Finally, the HORST key pair is used to sign the message. Which trees inside the hyper-tree are used (which in turn determines the WOTS⁺ key pairs used for the signature) and which HORST key pair is determined by the pseudorandomly generated index not shown here.

We use a simple addressing scheme for pseudorandom key generation. An address is a bit string of length $a = \lceil \log(d + 1) \rceil + (d - 1)(h/d) + (h/d) = \lceil \log(d + 1) \rceil + h$. The address of a WOTS⁺ key pair is obtained by encoding the layer of the tree it belongs to as a $\log(d+1)$ -bit string (using $d - 1$ for the top layer with a single tree). Then, appending the index of the tree in the layer encoded as a $(d - 1)(h/d)$ -bit string (we number the trees from left to right, starting with 0 for the left-most tree). Finally, appending the index of the WOTS⁺ key pair within the tree encoded as a (h/d) -bit string (again numbering from left to right, starting with 0). The address of the HORST key pair is obtained using the address of the WOTS⁺ key pair used to sign its public key and placing d as the layer value in the address string, encoded as $\lceil \log(d + 1) \rceil$ bit string. To give an example: In SPHINCS-256, an address needs 64 bits.

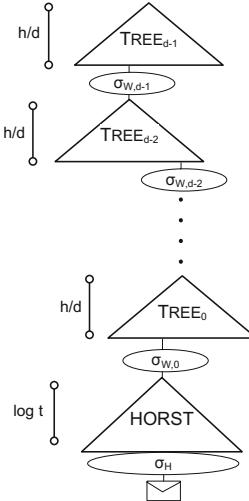


Fig. 3. Virtual structure of a SPHINCS signature

Key Generation Algorithm ($(\text{SK}, \text{PK}) \leftarrow \text{kg}(1^n)$): The key generation algorithm first samples two secret key values $(\text{SK}_1, \text{SK}_2) \in \{0, 1\}^n \times \{0, 1\}^n$. The value SK_1 is used for pseudorandom key generation. The value SK_2 is used to generate an unpredictable index in `sign` and pseudorandom values to randomize the message hash in `sign`. Also, p uniformly random n -bit values $\mathbf{Q} \xleftarrow{\$} \{0, 1\}^{p \times n}$ are sampled as bitmasks where $p = \max\{w-1, 2(h + \lceil \log \ell \rceil), 2 \log t\}$. These bitmasks are used for all WOTS⁺ and HORST instances as well as for the trees. In the following we use $\mathbf{Q}_{\text{WOTS}^+}$ for the first $w-1$ bitmasks (of length n) in \mathbf{Q} , $\mathbf{Q}_{\text{HORST}}$ for the first $2 \log t$, $\mathbf{Q}_{L-\text{Tree}}$ for the first $2\lceil \log \ell \rceil$, and \mathbf{Q}_{Tree} for the $2h$ strings of length n in \mathbf{Q} that follow $\mathbf{Q}_{L-\text{Tree}}$.

The remaining part of `kg` consists of generating the root node of the tree on layer $d-1$. Towards this end the WOTS⁺ key pairs for the single tree on layer $d-1$ are generated. The seed for the key pair with address $A = (d-1||0||i)$ where $i \in [2^{h/d}-1]$ is computed as $\mathcal{S}_A \leftarrow \mathcal{F}_a(A, \text{SK}_1)$, evaluating the PRF on input A with key SK_1 . In general, the seed for a WOTS⁺ key pair with address A is computed as $\mathcal{S}_A \leftarrow \mathcal{F}_a(A, \text{SK}_1)$ and we will assume from now on that these seeds are known to any algorithm that knows SK_1 . The WOTS⁺ public key is computed as $\text{pk}_A \leftarrow \text{WOTS}.\text{kg}(\mathcal{S}_A, \mathbf{Q}_{\text{WOTS}^+})$. The i th leaf L_i of the tree is the root of an L-Tree that compresses pk_A using bit masks $\mathbf{Q}_{L-\text{Tree}}$. Finally, a binary hash tree is built using the constructed leaves and its root node becomes PK_1 .

The SPHINCS secret key is $\text{SK} = (\text{SK}_1, \text{SK}_2, \mathbf{Q})$, the public key is $\text{PK} = (\text{PK}_1, \mathbf{Q})$. `kg` returns the key pair $((\text{SK}_1, \text{SK}_2, \mathbf{Q}), (\text{PK}_1, \mathbf{Q}))$.

Signature Algorithm ($\Sigma \leftarrow \text{sign}(M, \text{SK})$): On input of a message $M \in \{0, 1\}^*$ and secret key $\text{SK} = (\text{SK}_1, \text{SK}_2, \mathbf{Q})$, `sign` computes a randomized message digest $D \in \{0, 1\}^m$: First, a pseudorandom $R = (R_1, R_2) \in \{0, 1\}^n \times \{0, 1\}^n$ is computed as

$R \leftarrow \mathcal{F}(M, \mathbf{SK}_2)$. Then, $D \leftarrow \mathcal{H}(R_1, M)$ is computed as the randomized hash of M using the first n bits of R as randomness. The latter n bits of R are used to select a HORST keypair, computing an h bit index $i \leftarrow \text{CHOP}(R_2, h)$ as the first h bits of R_2 . Note, that signing is deterministic, i.e. we need no real randomness as all required 'randomness' is pseudorandomly generated using PRF \mathcal{F} .

Given index i , the HORST key pair with address $A_{\text{HORST}} = (d||i(0, (d-1)h/d)||i((d-1)h/d, h/d))$ is used to sign the message digest D , i.e., the first $(d-1)h/d$ bits of i are used as tree index and the remaining bits for the index within the tree. The HORST signature and public key $(\sigma_H, \mathbf{pk}_H) \leftarrow (D, \mathcal{S}_{A_{\text{HORST}}}, \mathbf{Q}_{\text{HORST}})$ are computed using the HORST bitmasks and the seed $\mathcal{S}_{A_{\text{HORST}}} \leftarrow \mathcal{F}_a(A_{\text{HORST}}, \mathbf{SK}_1)$.

The SPHINCS signature $\Sigma = (i, R_1, \sigma_H, \sigma_{W,0}, \mathbf{Auth}_{A_0}, \dots, \sigma_{W,d-1}, \mathbf{Auth}_{A_{d-1}})$ contains besides index i , randomness R_1 and HORST signature σ_H also one WOTS⁺ signature and one authentication path $\sigma_{W,i}, \mathbf{Auth}_{A_i}, i \in [d-2]$ per layer. These are computed as follows: The WOTS⁺ key pair with address A_0 is used to sign \mathbf{pk}_H , where A_0 is the address obtained taking A_{HORST} and setting the first $\lceil \log(d+1) \rceil$ bits to zero. This is done running $\sigma_{W,1} \leftarrow (\mathbf{pk}_H, \mathcal{S}_{A_0}, \mathbf{Q}_{\text{WOTS}^+})$ using the WOTS⁺ bitmasks. Then the authentication path $\mathbf{Auth}_{i((d-1)h/d, h/d)}$ of the used WOTS⁺ key pair is computed. Next, the WOTS⁺ public key $\mathbf{pk}_{W,0}$ is computed running $\mathbf{pk}_{W,0} \leftarrow \text{WOTS.vf}(\mathbf{pk}_H, \sigma_{W,0}, \mathbf{Q}_{\text{WOTS}^+})$. The root node ROOT_0 of the tree is computed by first compressing $\mathbf{pk}_{W,0}$ using an L-Tree. Then Algorithm 1 is applied using the index of the WOTS⁺ key pair within the tree, the root of the L-Tree and $\mathbf{Auth}_{i((d-1)h/d, h/d)}$.

This procedure gets repeated for layers 1 to $d-1$ with the following two differences. On layer $1 \leq j < d$, WOTS⁺ is used to sign ROOT_{j-1} , the root computed at the end of the previous iteration. The address of the WOTS⁺ key pair used on layer j is computed as $A_j = (j||i(0, (d-1-j)h/d)||i((d-1-j)h/d, h/d))$, i.e. on each layer the last (h/d) bits of the tree address become the new leaf address and the remaining bits of the former tree address become the new tree address.

Finally, `sign` outputs $\Sigma = (i, R_1, \sigma_H, \sigma_{W,0}, \mathbf{Auth}_{A_0}, \dots, \sigma_{W,d-1}, \mathbf{Auth}_{A_{d-1}})$.

Verification Algorithm ($b \leftarrow \text{vf}(M, \Sigma, \mathbf{PK})$): On input of a message $M \in \{0, 1\}^*$, a signature Σ , and a public key \mathbf{PK} , the algorithm computes the message digest $D \leftarrow \mathcal{H}(R_1, M)$ using the randomness R_1 contained in the signature. The message digest D and the HORST bitmasks $\mathbf{Q}_{\text{HORST}}$ from \mathbf{PK} are used to compute the HORST public key $\mathbf{pk}_H \leftarrow \text{HORST.vf}(D, \sigma_H, \mathbf{Q}_{\text{HORST}})$ from the HORST signature. If HORST.vf returns fail, verification returns false. The HORST public key in turn is used together with the WOTS⁺ bit masks and the WOTS⁺ signature to compute the first WOTS⁺ public key $\mathbf{pk}_{W,0} \leftarrow \text{WOTS.vf}(\mathbf{pk}_H, \sigma_{W,0}, \mathbf{Q}_{\text{WOTS}^+})$. An L-Tree is used to compute $L_{i((d-1)h/d, h/d)}$, the leaf corresponding to $\mathbf{pk}_{W,0}$. Then, the root ROOT_0 of the respective tree is computed using Algorithm 1 with index $i((d-1)h/d, h/d)$, leaf $L_{i((d-1)h/d, h/d)}$ and authentication path \mathbf{Auth}_0 .

Then, this procedure gets repeated for layers 1 to $d-1$ with the following two differences. First, on layer $1 \leq j < d$ the root of the previously processed

tree ROOT_{j-1} is used to compute the WOTS⁺ public key $\text{pk}_{W,j}$. Second, the leaf computed from $\text{pk}_{W,j}$ using an L-Tree is $L_{i((d-1-j)h/d,h/d)}$, i.e., the index of the leaf within the tree can be computed cutting off the last $j(h/d)$ bits of i and then using the last (h/d) bits of the resulting bit string.

The result of the final repetition on layer $d - 1$ is a value ROOT_{d-1} for the root node of the single tree on the top layer. This value is compared to the first element of the public key, i.e., $\text{PK}_1 \stackrel{?}{=} \text{ROOT}_{d-1}$. If the comparison holds, vf returns **true**, otherwise it returns **false**.

Theoretical Performance. In the following we give rough theoretical performance values. We count runtimes counting the number of PRF, PRG and hash evaluations without distinguishing the different PRFs, PRGs, and hashes.

Sizes: A SPHINCS secret key consists of two n bit seeds and the $p = \max\{w - 1, 2(h + \lceil \log \ell \rceil), 2 \log t\} n$ bit bitmasks, summing up to $(2 + p)n$ bits. The public key contains a single n bit hash and the bitmasks: $(1 + p)n$ bits. A signature contains one h bit index and n bits of randomness. Moreover, it contains a HORST signature ($((k(\log t) - x + 1) + 2^x)n$ bits), d WOTS signatures (ℓn bits each), and a total of h authentication path nodes (n bits each). This gives a signature size of $((k(\log t) - x + 1) + 2^x) + d\ell + h + 1 n + h$ bits.

Runtimes: SPHINCS key generation consists of building the top tree. This takes for leaf generation $2^{h/d}$ times the following: One PRF call, one PRG call, one WOTS⁺ key generation (ℓw hashes), and one L-Tree ($\ell - 1$ hashes). Building the tree adds another $2^{h/d} - 1$ hashes. Together these are $2^{h/d}$ PRF and PRG calls and $(\ell(w + 1))2^{h/d} - 1$ hashes. Signing requires one PRF call to generate the index and the randomness for the message hash as well as the message hash itself. Then one PRF call to generate a HORST seed and a HORST signature. In addition, d trees have to be built, adding d times the time for key generation. The WOTS⁺ signatures can be extracted while running WOTS⁺ key generation, hence they add no extra cost. This sums up to $d2^{h/d} + 2$ PRF calls, $d2^{h/d} + 1$ PRG calls, and $2t + d((\ell(w + 1))2^{h/d} - 1)$ hashes. Finally, verification needs the message hash, one HORST verification, and d times a WOTS⁺ verification ($< \ell w$ hashes), computing an L-Tree, and $h/d - 1$ hashes to compute the root. This leads to a total of $k(\log t) - x + 1 + 2^x + d(\ell(w + 1) - 2) + h$ hashes.

3 Security Analysis

We now discuss the security of SPHINCS. We first give a reduction from standard hash function properties. Afterwards we discuss the best generic attacks on these properties using quantum computers. Definitions of the used properties can be found in Appendix A. For our security analysis we group the message hash and the mapping used within HORST to a function $\mathcal{H}_{k,t}$ that maps bit strings of arbitrary length to a subset of $\{0, \dots, t - 1\}$ with at most k elements.

3.1 Security Reduction

We will now prove our main theorem which states that SPHINCS is secure as long as the used function (families) provide certain standard security properties. These properties are fulfilled by secure cryptographic hash functions, even against quantum attacks. For the exact statement in the proof we use the notion of insecurity functions. An insecurity function $\text{InSec}^p(s; t, q)$ describes the maximum success probability of any adversary against property p of primitive s , running in time $\leq t$, and (if given oracle access, e.g. to a signing oracle) making no more than q queries. To avoid the non-constructive high-probability attacks discussed in [11], we measure time with the *AT* metric rather than the RAM metric. Properties are one-wayness (ow), second-preimage resistance (SPR), undetectability (UD), secure pseudorandom generator (PRG), secure pseudorandom function family (PRF), and γ -subset resilience (γ -sr).

Theorem 1. *SPHINCS is existentially unforgeable under q_s -adaptive chosen message attacks if*

- F is a second-preimage resistant, undetectable one-way function,
- H is a second-preimage resistant hash function,
- G_λ is a secure pseudorandom generator for values $\lambda \in \{\ell, t\}$,
- \mathcal{F}_λ is a pseudorandom function family for $\lambda = a$,
- \mathcal{F} is a pseudorandom function family, and
- for the subset-resilience of $\mathcal{H}_{k,t}$ it holds that

$$\sum_{\gamma=1}^{\infty} \min \left\{ 2^{\gamma(\log q_s - h) + h}, 1 \right\} \cdot \text{Succ}_{\mathcal{H}_{k,t}}^{\gamma\text{-sr}}(\mathcal{A}) = \text{negl}(n)$$

for any probabilistic polynomial-time adversary \mathcal{A} , where $\text{Succ}_{\mathcal{H}_{k,t}}^{\gamma\text{-sr}}(\mathcal{A})$ denotes the success probability of \mathcal{A} against the γ -subset resilience of $\mathcal{H}_{k,t}$.

More specifically, the insecurity function $\text{InSec}^{\text{EU-CMA}}(\text{SPHINCS}; \xi, q_s)$ describing the maximum success probability of all adversaries against the existential unforgeability under q_s -adaptive chosen message attacks, running in time $\leq \xi$, is bounded by

$$\begin{aligned} & \text{InSec}^{\text{EU-CMA}}(\text{SPHINCS}; \xi, q_s) \\ & \leq \text{InSec}^{\text{PRF}}(\mathcal{F}; \xi, q_s) + \text{InSec}^{\text{PRF}}(\mathcal{F}_a; \xi, \#_{fts} + \#_{ots}) \\ & + \#_{ots} \cdot \text{InSec}^{\text{PRG}}(G_\ell; \xi) + \#_{fts} \cdot \text{InSec}^{\text{PRG}}(G_t; \xi) \\ & + \#_{tree} \cdot 2^{h/d + \lceil \log \ell \rceil} \cdot \text{InSec}^{\text{SPR}}(H; \xi) \\ & + \#_{ots} \cdot (\ell w^2 \cdot \text{InSec}^{\text{UD}}(F; \xi) + \ell w \cdot \text{InSec}^{\text{OW}}(F; \xi) + \ell w^2 \cdot \text{InSec}^{\text{SPR}}(F; \xi)) \\ & + \#_{fts} \cdot 2t \cdot \text{InSec}^{\text{SPR}}(H; \xi) + \#_{fts} \cdot t \cdot \text{InSec}^{\text{OW}}(F; \xi) \\ & + \sum_{\gamma=1}^{\infty} \min \left\{ 2^{\gamma(\log q_s - h) + h}, 1 \right\} \cdot \text{InSec}^{\gamma\text{-sr}}(\mathcal{H}_{k,t}; \xi), \end{aligned}$$

where $\#_{ots} = \min \left\{ \sum_{i=1}^d 2^{ih/d}, dq_s \right\}$ denotes the maximum number of WOTS⁺ key pairs, $\#_{fts} = \min \{2^h, q_s\}$ denotes the maximum number of HORST key pairs, and $\#_{tree} = \min \left\{ \sum_{i=1}^{d-1} 2^{ih/d}, (d-1)q_s \right\}$ denotes the maximum number of subtrees used answering q_s signature queries.

Before we give the proof, note that although there is a bunch of factors within the exact security statement, the reduction is tight. All the factors are constant / independent of the security parameter. They arise as all the primitives are used many times. E.g., the pseudorandom generator G_ℓ is used for every WOTS⁺ key pair and an adversary can forge a signature if it can distinguish the output for one out of the $\#_{ots}$ applications of G_ℓ from a random bit string. Similar explanations exist for the other factors.

Proof. In the following we first show that the success probability of any probabilistic polynomial-time adversary \mathcal{A} that attacks the EU-CMA security of SPHINCS is negligible in the security parameter. Afterwards, we analyze the exact success probability of \mathcal{A} and show that it indeed fulfills the claimed bound. First consider the following six games:

Game 1 is the original EU-CMA game against SPHINCS.

Game 2 differs from Game 1 in that the value R used to randomize the message hash and to choose the index i is chosen uniformly at random instead of using \mathcal{F} .

Game 3 is similar to Game 2 but this time all used WOTS⁺ and HORST seeds are generated uniformly at random and stored in some list for reuse instead of generating them using \mathcal{F}_a .

Game 4 is similar to Game 3 but this time no pseudorandom key generation is used inside WOTS⁺. Instead, all WOTS⁺ secret key elements are generated uniformly at random and stored in some list for reuse.

Game 5 is similar to Game 4 but this time no pseudorandom key generation is used at all. Instead, also all HORST secret key elements are generated uniformly at random and stored in some list for reuse.

The difference in the success probability of \mathcal{A} between playing Game 1 and Game 2 must be negligible. Otherwise we could use \mathcal{A} as an distinguisher against the pseudorandomness of \mathcal{F} . Similarly, the difference in the success probability of \mathcal{A} between playing Game 2 and Game 3 must be negligible. Otherwise, we could use \mathcal{A} as an distinguisher against the pseudorandomness of \mathcal{F}_a . Also the difference in the success probability of \mathcal{A} between playing Game 3 and Game 4 and playing Game 4 and Game 5 must be negligible. Otherwise, \mathcal{A} could be used to distinguish the outputs of the PRG G_ℓ (resp. G_t) from uniformly random bit strings.

It remains to limit the success probability of \mathcal{A} running in Game 5. Assume that \mathcal{A} makes q_s queries to the signing oracle before outputting a valid forgery

$$M^*, \Sigma^* = (i^*, R^*, \sigma_H^*, \sigma_{W,0}^*, \text{Auth}_{A_0}^*, \dots, \sigma_{W,d-1}^*, \text{Auth}_{A_{d-1}}^*).$$

The index i^* was used to sign at least one of the query messages with overwhelming probability. Otherwise, \mathcal{A} could be turned into a (second-)preimage finder for H that succeeds with non-negligible probability. Hence, we assume from now on i^* was used before. While running $\text{vf}(M^*, \Sigma^*, \text{PK})$ we can extract the computed HORST public key pk_H^* as well as the computed WOTS⁺ public keys $\text{pk}_{W,j}^*$ and the root nodes of the trees containing these WOTS⁺ public keys ROOT_j^* for all levels $j \in [d - 1]$. In addition, we compute the respective values pk_H , $\text{pk}_{W,j}$ and ROOT_j using the list of secret key elements. All required elements must be contained in the lists as i^* was used before.

Next we compare these values in reverse order of computation, i.e., starting with $\text{pk}_{W,d-1} \stackrel{?}{=} \text{pk}_{W,d-1}^*$, then $\text{ROOT}_{d-2} \stackrel{?}{=} \text{ROOT}_{d-2}^*$, and so forth. Then one of the following four mutually exclusive cases must appear:

Case 1: The first occurrence of a difference happens for a WOTS⁺ public key.

As shown in [18] this can only happen with negligible probability. Otherwise, we can use \mathcal{A} to compute second-preimages for H with non-negligible success probability.

Case 2: The first difference occurs for two root nodes $\text{ROOT}_j \neq \text{ROOT}_j^*$. This implies a forgery for the WOTS⁺ key pair used to sign ROOT_j . As shown in [26] this can only happen with negligible advantage. Otherwise, we could use \mathcal{A} to either break the one-wayness, the second-preimage resistance, or the undetectability of F with non-negligible success probability.

Case 3: The first spotted difference is two different HORST public keys. As for Case 2, this implies a WOTS⁺ forgery and can hence only appear with negligible probability.

Case 4: All the public keys and root nodes are equal, i.e. no difference occurs.

We excluded all cases but Case 4 which we analyze now. The analysis consists of a sequence of mutually exclusive cases. Recall that the secret key elements for this HORST key pair are already fixed and contained in the secret value list as i^* was used in the query phase. First, we compare the values of all leaf nodes that can be derived from σ_H^* with the respective values derived from the list entries. These are the hashes of the secret key elements in the signature and the authentication path nodes for level 0. The case that there exists a difference can only appear with negligible probability, as otherwise \mathcal{A} could be used to compute second-preimages for H with non-negligible probability following the proof in [18]. Hence, we assume from now on all of these are equal.

Second, the indices of the secret key values contained in σ_H^* have either all been published as parts of query signatures or at least one index has not been published before. The latter case can only appear with negligible probability. Otherwise, \mathcal{A} could be turned into a preimage finder for F that has non-negligible success probability. Finally, we can limit the probability that all indices have been published as parts of previous signatures.

Recall, when computing the signatures on the query messages, the indices were chosen uniformly at random. Hence, the probability that a given index reoccurs γ times, i.e., is used for γ signatures, is equal to the probability of the

event $C(2^h, q_s, \gamma)$ that after q_s samples from a set of size 2^h at least one value was sampled γ times. This probability can in turn be bound by

$$\Pr[C(2^h, q_s, \gamma)] \leq \frac{1}{2^{h(\gamma-1)}} \binom{q_s}{\gamma} \leq \frac{q_s^\gamma}{2^{h(\gamma-1)}} = 2^{\gamma(\log q_s - h) + h} \quad (1)$$

as shown in [36]. Using Equation (1), the probability for this last case can be written as

$$\sum_{\gamma=1}^{\infty} \min \left\{ 2^{\gamma(\log q_s - h) + h}, 1 \right\} \cdot \text{Succ}_{\mathcal{H}_{k,t}}^{\gamma-\text{sr}}(\mathcal{A}),$$

i.e. the sum over the probabilities that there exists at least one index that was used γ times multiplied by \mathcal{A} 's success probability against the γ -subset resilience of $\mathcal{H}_{k,t}$. This sum is negligible per assumption. Hence the success probability of \mathcal{A} is negligible which concludes the asymptotic proof.

Probabilities. Now we take a look at the exact success probability $\epsilon = \text{Succ}_{\text{SPHINCS}}^{\text{EU-CMA}}(\mathcal{A})$ of an adversary \mathcal{A} that runs in time ξ and makes q_s signature queries. Per definition, \mathcal{A} 's probability of winning Game 1 is ϵ . In what follows let $\#_{ots} = \min \left\{ \sum_{i=1}^d 2^{ih/d}, dq_s \right\}$ denote the maximum number of WOTS⁺ key pairs, $\#_{fts} = \min \left\{ 2^h, q_s \right\}$ the maximum number of HORST key pairs, and $\#_{tree} = \min \left\{ \sum_{i=1}^{d-1} 2^{ih/d}, (d-1)q_s \right\}$ the maximum number of subtrees used while answering signature queries. Now, from the definition of the insecurity functions we get that the differences in the success probabilities of \mathcal{A} playing two neighboring games from the above series of games are bounded by $\text{InSec}^{\text{PRF}}(\mathcal{F}; \xi, q_s)$, $\text{InSec}^{\text{PRF}}(\mathcal{F}_a; \xi, \#_{fts} + \#_{ots})$, $\#_{ots} \cdot \text{InSec}^{\text{PRG}}(\mathcal{G}_\ell; \xi)$, $\#_{fts} \cdot \text{InSec}^{\text{PRG}}(\mathcal{G}_t; \xi)$, respectively. Hence, ϵ is bounded by \mathcal{A} 's probability of winning Game 5 plus the sum of the above bounds.

It remains to limit \mathcal{A} 's success probability in Game 5. A more detailed analysis shows that the case that i^* was not used before is also covered by the following cases. (The reason is that at some point the path from the message to the root must meet a path which was used in the response to a query before.) So we only have to consider the four cases. The probability that \mathcal{A} succeeds with a Case 1 forgery is limited by $\#_{tree} \cdot 2^{h/d + \lceil \log \ell \rceil} \cdot \text{InSec}^{\text{SPR}}(\mathcal{H}; \xi)$. This bound can be obtained by first guessing a tree and then following the proof in [18]. The combined probability that \mathcal{A} succeeds with a Case 2 or Case 3 forgery is limited by $\#_{ots} \cdot \text{InSec}^{\text{EU-CMA}}(\text{WOTS}^+; t, 1) \leq \#_{ots} \cdot (\ell w^2 \cdot \text{InSec}^{\text{UD}}(\mathcal{F}; \xi) + \ell w \cdot \text{InSec}^{\text{OW}}(\mathcal{F}; \xi) + \ell w^2 \cdot \text{InSec}^{\text{SPR}}(\mathcal{F}; \xi))$. Similarly to the last case, this bound can be obtained by first guessing the WOTS⁺ key pair \mathcal{A} will forge a signature for and then following the proof from [26]².

Case 4 consists of another three mutually exclusive cases. The probability that \mathcal{A} succeeds by inserting new leaves into the HORST tree can be bounded by $\#_{fts} \cdot 2t \cdot \text{InSec}^{\text{SPR}}(\mathcal{H}; \xi)$. This can be seen, first guessing the HORST key pair and then following again the proof in [18]. The probability that \mathcal{A} succeeds by providing a valid value for an index not included in previous signatures can be bounded

² The used bound is actually an improved bound from [25].

by $\#_{fts} \cdot t \cdot \text{InSec}^{\text{OW}}(F; \xi)$. This can be shown, first guessing the HORST key pair and afterwards, guessing the index. Finally, the probability that \mathcal{A} succeeds finding a message for which it already knows the values to be opened from previous signatures can be bounded by $\sum_{\gamma=1}^{\infty} \min\{2^{\gamma(\log q_s - h) + h}, 1\} \cdot \text{InSec}^{\gamma\text{-sr}}(\mathcal{H}_{k,t}; \xi)$.

As the three cases are mutually exclusive, the probability that \mathcal{A} succeeds with a Case 4 forgery is bound by the sum of the three probabilities above. Similarly, the probability of \mathcal{A} winning Game 5 is bound by the sum of the probabilities of \mathcal{A} succeeding in Case 1 - 4. This finally leads the claimed bound. \square

3.2 Generic Attacks

As a complement to the above reduction, we now analyze the concrete complexity of various attacks, both pre-quantum and post-quantum. Recall that $\mathcal{H}_{k,t}(R, M)$ applied to message $M \in \{0, 1\}^*$ and randomness $R \in \{0, 1\}^n$ works as follows. First, the message digest is computed as $M' = \mathcal{H}(R, M) \in \{0, 1\}^m$. Then, M' is split into k bit strings, each of length $\log t$. Finally, each of these bit strings is interpreted as an unsigned integer. Thus, the output of $\mathcal{H}_{k,t}$ is an ordered subset of k values out of the set $[t - 1]$ (possibly with repetitions).

Subset-Resilience. The main attack vector against SPHINCS is targeting subset-resilience. The obvious first attack is to simply replace (R, M) in a valid signature with (R', M') , hoping that $\mathcal{H}_{k,t}(R, M) = \mathcal{H}_{k,t}(R', M')$. This violates strong unforgeability if $(R, M) \neq (R', M')$, and it violates existential unforgeability if $M \neq M'$. Finding a second preimage of (R, M) under $\mathcal{H}_{k,t}$ costs 2^m pre-quantum but only $2^{m/2}$ post-quantum (Grover's algorithm). To reach success probability p takes time $\sqrt{p}2^{m/2}$.

The attacker does succeed in reusing $\mathcal{H}_{k,t}(R, M)$ if $\mathcal{H}_{k,t}(R', M')$ contains the same indices as $\mathcal{H}_{k,t}(R, M)$, because then he can permute the HORST signature for (R, M) accordingly to obtain the HORST signature for (R', M') . If k^2 is considerably smaller than t then the k indices in a hash are unlikely to contain any collisions, so there are about $2^m/k!$ equivalence classes of hashes under permutations. It is easy to map each hash to a numerical representative of its equivalence class, effectively reducing the pre-quantum second-preimage cost from 2^m to $2^m/k!$, and the post-quantum second-preimage cost from $2^{m/2}$ to $\sqrt{2^m/k!}$.

More generally, when γ valid signatures use the same HORST key, the attacker can mix and match the HORST signatures. All the attacker needs is to break γ -subset-resilience: i.e., find $\mathcal{H}_{k,t}(R', M')$ so that the set of k indices in it is a subset of the union of the indices in the γ valid signatures. The union has size about γk (at most γk , and usually close to γk if γk is not very large compared to t), so a uniform random number has probability about $\gamma k/t$ of being in the union, and if the k indices were independent uniform random numbers then they would have probability about $(\gamma k)^k/t^k$ of all being in the union. The expected cost of a pre-quantum attack is about $t^k/(\gamma k)^k$, and the expected cost of a post-quantum attack is about $t^{k/2}/(\gamma k)^{k/2}$.

Of course, this attack cannot start unless the signer in fact produced γ valid signatures using the same HORST key. After a total of q signatures, the

probability of any particular HORST key being used exactly γ times is exactly $\binom{q}{\gamma}(1 - 1/2^h)^{q-\gamma}(1/2^h)^\gamma$. This probability is bounded above by $(q/2^h)^\gamma$ in the above proof; a much tighter approximation is $(q/2^h)^\gamma \exp(-q/2^h)/\gamma!$. If the ratio $\rho = q/2^h$ is significantly smaller than 1 then there will be approximately q keys used once, approximately $\rho q/2$ keys used twice, approximately $\rho^2 q/6$ keys used three times, and so on. The chance that some key will be used γ times, for $\gamma = h/\log(1/\rho) + \delta$, is approximately $\rho^\delta/\gamma!$.

For example, consider $t = 2^{16}$, $k = 32$, $h = 60$, and $q = 2^{50}$. There is a noticeable chance, approximately $2^{-9.5}$, that some HORST key is used 6 times. For $\gamma = 6$ the expected cost of a pre-quantum attack is 2^{269} and the expected cost of a post-quantum attack is 2^{134} . Increasing γ to 9 reduces the post-quantum cost below 2^{128} , specifically to $2^{125.3}$, but the probability of a HORST key being used 9 times is below 2^{-48} . Increasing γ to 10, 11, 12, 13, 14, 15 reduces the cost to $2^{122.8}, 2^{120.6}, 2^{118.6}, 2^{116.8}, 2^{115.1}, 2^{113.5}$ respectively, but reduces the probability to $2^{-61}, 2^{-75}, 2^{-88}, 2^{-102}, 2^{-116}, 2^{-130}$ respectively.

Security degrades as q grows closer to 2^h . For example, for $q = 2^{60}$ the attacker finds $\gamma = 26$ with probability above 2^{-30} , and then a post-quantum attack costs only about 2^{100} . Of course, the signer is in control of the number of messages signed: for example, even if the signer's key is shared across enough devices to continuously sign 2^{20} messages per second, signing 2^{50} messages would take more than 30 years.

One-Wayness. The attacker can also try to work backwards from a hash output to an n -bit hash input that was not revealed by the signer (or a n -bit half of a $2n$ -bit hash input where the other half was revealed by the signer). If the hash inputs were independent uniform random n -bit strings then this would be a standard preimage problem; generic pre-quantum preimage search costs 2^n , and generic post-quantum preimage search (again Grover) costs $2^{n/2}$.

The attacker can also merge preimage searches for n -bit-to- n -bit hashes. (For $2n$ -bit-to- n -bit hashes the known n input bits have negligible chance of repeating.) For pre-quantum attacks the cost of generic T -target preimage attacks is well known to drop by a factor of T ; here T is bounded by approximately 2^h (the exact bound depends on q), for a total attack cost of approximately 2^{n-h} . For post-quantum attacks, it is well known that $2^{n/2}/\sqrt{T}$ quantum *queries* are necessary and sufficient for generic T -target preimage attacks (assuming $T < 2^{n/3}$), but there is overhead beyond the queries. An analysis of this overhead by Bernstein [8] concludes that all known post-quantum collision-finding algorithms cost at least $2^{n/2}$, implying that the post-quantum cost of multi-target preimage attacks is also $2^{n/2}$. For example, for $n = 256$ and $T = 2^{56}$ the best post-quantum attacks use only 2^{100} queries but still cost 2^{128} .

Second-Preimage Resistance. As for the message hash, finding a second preimage of either a message $M \in \{0,1\}^n$ under F or a message $\mathcal{M} \in \{0,1\}^{2n}$ under H costs 2^n pre-quantum and $2^{n/2}$ post-quantum (Grover's algorithm).

PRF, PRG, and Undetectability. The hash inputs are actually obtained from a chain of PRF outputs, PRG outputs, and lower-level hash outputs. The

attacker can try to find patterns in these inputs, for example by guessing the PRF key, the PRG seed, or an input to F that ends up at a target value after a number of rounds of the chaining function. All generic attacks again cost 2^n pre-quantum and $2^{n/2}$ post-quantum.

4 SPHINCS-256

In addition to the general construction of SPHINCS, we propose a specific instantiation called SPHINCS-256. The parameters, functions, and resulting key and signature sizes of SPHINCS-256 are summarized in Table 1. This section describes how these parameters and functions were chosen.

Parameters. The parameters for SPHINCS-256 were selected with two goals in mind: (1) long-term 2^{128} security against attackers with access to quantum computers; (2) a good tradeoff between speed and signature size. The first goal determined the security parameter $n = 256$, which in turn determined the name SPHINCS-256. Optimizing the remaining parameters required deciding on the relative importance of speed and signature size. After searching a large parameter space we settled on the parameters $m = 512$, $h = 60$, $d = 12$, $w = 16$, $t = 2^{16}$, and $k = 32$, implying $\ell = 67$, $x = 6$, and $a = 64$. These choices are small enough and fast enough for a wide range of applications. Of course, one can also define different SPHINCS instantiations, changing the remaining parameters in favor of either speed or signature size.

Security of SPHINCS-256. SPHINCS-256 uses $n = 256$, $m = 512$, $h = 60$, $d = 12$, $w = 16$, $t = 2^{16}$, $k = 32$ as parameters. Hence, considering attackers that have access to a large scale quantum computer this means the following. Assuming the best attacks against the used hash functions are generic attacks as described in the last section, $\mathcal{H}_{k,t}$ provides security above 2^{128} regarding subset-resilience, F and H provide 2^{128} security against preimage, second-preimage and in case of F undetectability attacks. Similarly, the used PRFs and PRGs provide security 2^{128} . Summing up, SPHINCS-256 provides 2^{128} security against post-quantum attackers under the assumptions above.

Fast Fixed-Size Hashing. The primary cost metric in the literature on cryptographic hash functions, for example in the recently concluded SHA-3 competition, is performance for long inputs. However, what is most important for SPHINCS and hash-based signatures in general is performance for short inputs. The hashing in SPHINCS consists primarily of applying F to n -bit inputs and secondarily of applying H to $2n$ -bit inputs.

Short-input performance was emphasized in a recent MAC/PRF design [1] from Aumasson and Bernstein. We propose short-input performance as a similarly interesting target for hash-function designers.

Designing a new hash function is not within the scope of this paper: we limit ourselves to evaluating the short-input performance of previously designed components that appear to have received adequate study. Below we explain our

Table 1. SPHINCS-256 parameters and functions for the 128-bit post-quantum security level and resulting signature and key sizes

Parameter	Value	Meaning
n	256	bitlength of hashes in HORST and WOTS
m	512	bitlength of the message hash
h	60	height of the hyper-tree
d	12	layers of the hyper-tree
w	16	Winternitz parameter used for WOTS signatures
t	2^{16}	number of secret-key elements of HORST
k	32	number of revealed secret-key elements per HORST sig.
Functions		
Hash \mathcal{H} :	$\mathcal{H}(R, M) = \text{BLAKE-512}(R \ M)$	
PRF \mathcal{F}_a :	$\mathcal{F}_a(A, K) = \text{BLAKE-256}(K \ A)$	
PRF \mathcal{F} :	$\mathcal{F}(M, K) = \text{BLAKE-512}(K \ M)$	
PRG G_λ :	$G_\lambda(\text{SEED}) = \text{ChaCha12}_{\text{SEED}}(0 _{0,\dots,\lambda-1})$	
Hash F:	$F(M_1) = \text{CHOP}(\pi_{\text{ChaCha}}(M_1 \ C), 256)$	
Hash H:	$H(M_1 \ M_2) = \text{CHOP}(\pi_{\text{ChaCha}}(\pi_{\text{ChaCha}}(M_1 \ C) \oplus (M_2 \ 0^{256})), 256)$	
Sizes		
Signature size:	41000 bytes	
Public-key size:	1056 bytes	
Private-key size:	1088 bytes	

selection of specific functions $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$ and $H : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ for $n = 256$.

Review of Permutation-Based Cryptography. Rivest suggested strengthening the DES cipher by “whitening” the input and output: i.e., encrypting a block M under key (K, K_1, K_2) as $E_K(M \oplus K_1) \oplus K_2$, where E_K means DES using key K . Even and Mansour [21] suggested eliminating the original key K : i.e., encrypting a block M under key (K_1, K_2) as $E(M \oplus K_1) \oplus K_2$, where E is an *unkeyed* public permutation. Kilian and Rogaway [28, Section 4] suggested taking $K_1 = K_2$.

Combining all of these suggestions means encrypting M under key K as $E(M \oplus K) \oplus K$; see, e.g., [29], [7], and [20]. Trivial 0-padding or, more generally, padding with a constant allows M and K to be shorter than the block length of E : for example, the “Salsa20” cipher from [7] actually produces $E(K, M, C) + (K, M, C)$, where C is a constant. The PRF security of Salsa20 is tightly equivalent to the PRF security of $E(K, M, C) + (K, 0, 0)$, which in turn implies the PRF security of the “HSalsa20” stream cipher [9] obtained by truncating $E(K, M, C)$.

Bertoni, Daemen, Peeters, and Van Assche [12] proposed building cryptographic hash functions from unkeyed permutations, and later proposed a specific “Keccak” hash function. The “sponge” construction used in [12], and in Keccak, hashes a $(b-c)$ -bit message K_1 to a $(b-c)$ -bit truncation of $E(K_1, C)$, where C is a c -bit constant; hashes a $2(b-c)$ -bit message (K_1, K_2) to a $(b-c)$ -bit truncation

of $E(E(K_1, C) \oplus (K_2, 0))$; etc. Sponges have been reused in many subsequent designs and studied in many papers. We ended up selecting the sponge structure for both F and H.

Note that the single-block hash here, a truncation of $E(K_1, C)$, is the same as an encryption of a constant nonce using a truncated- $E(K_1, M, C)$ cipher. Of course, there is no logical connection between the PRF security of this cipher and (e.g.) second-preimage resistance, but designers use the same techniques to build E for either context: consider, for example, the reuse of the Salsa20 permutation in the “Rumba20” [5] compression function, the reuse of a tweaked version of the “ChaCha20” permutation [6] in the “BLAKE” and “BLAKE2” [4] hash functions, and the reuse of the Keccak permutation in the “Keyak” [14] authenticated-encryption scheme.

Many other hash-function designs use input blocks as cipher keys, but in most cases the underlying ciphers use complicated “key schedules” rather than wrapping simple key addition around an unkeyed permutation. Both [7] and [13] state reasons to believe that unkeyed permutations provide the best performance-security tradeoff. Performance obviously played a large role in the selection of Salsa20/12 (Salsa20 reduced to 12 rounds) for the eSTREAM portfolio, the deployment of ChaCha20 in TLS [31], and the selection of Keccak as SHA-3. We did not find any non-permutation-based hash-function software competitive in performance with the permutation that we selected.

Choice of Permutation for $n = 256$. A sponge function using a b -bit permutation E and a c -bit “capacity” takes $b - c$ bits in each input block and produces $b - c$ bits of output. We require $b - c \geq 256$ so that a single call to E hashes 256 bits to 256 bits (and two calls to E hash 512 bits to 256 bits). The attacker can compute preimages by guessing the c missing bits and applying E^{-1} , so we also require $c \geq 256$.

We considered using the Keccak permutation, which has $b = 1600$, but this is overkill: it takes as long to hash a 256-bit block as it does to hash a 1000-bit block. There is a scaled-down version of Keccak with $b = 800$, but this is not part of SHA-3, and we do not know how intensively it has been analyzed.

After considering various other permutations we settled on ChaCha, which has $b = 512$. ChaCha is a slightly modified version of Salsa, advertising faster diffusion and at the same time better performance. The best key-recovery attacks known are from Aumasson, Fischer, Khazaei, Meier, and Rechberger [2] and are slightly faster than 2^{256} operations against 8 rounds of Salsa and 7 rounds of ChaCha, supporting the security advertisement. The eSTREAM portfolio recommends 12 rounds of Salsa20 as having a “comfortable margin for security” so we selected 12 rounds of ChaCha (ChaCha12). The Salsa and ChaCha permutations are not designed to simulate ideal permutations: they are designed to simulate ideal permutations with certain symmetries, i.e., ideal permutations of the orbits of the state space under these symmetries. The Salsa and ChaCha stream ciphers add their inputs to only part of the block and specify the rest of the block as asymmetric constants, guaranteeing that different inputs lie in different orbits. For the same reason we specify an asymmetric constant for C .

Specifically, let $\pi_{\text{ChaCha}} : \{0, 1\}^{512} \rightarrow \{0, 1\}^{512}$ denote the ChaCha12 permutation, let C be the bytes of the ASCII representation of “expand 32-byte to 64-byte state!” and let $\text{CHOP}(M, i)$ be the function that returns the first i bits of the string M . Then we define

$$\begin{aligned} F(M_1) &= \text{CHOP}(\pi_{\text{ChaCha}}(M_1 \| C), 256), \text{ and} \\ H(M_1 \| M_2) &= \text{CHOP}(\pi_{\text{ChaCha}}(\pi_{\text{ChaCha}}(M_1 \| C) \oplus (M_2 \| 0^{256})), 256). \end{aligned}$$

for any 256-bit strings M_1, M_2 .

Other Functions. We also use ChaCha12 directly for the PRG G_λ . Specifically, we define $G_\lambda(\text{SEED}) = \text{ChaCha12}_{\text{SEED}}(0)_0, \dots, \lambda-1$, i.e., we run ChaCha12 with key SEED and initialization vector 0 and take the first λ output bits.

For message hashing we use BLAKE, whose security was extensively studied as part of the SHA-3 competition. We also use BLAKE for the n -bit-output PRF and for the $2n$ -bit-output PRF: We define $\mathcal{H}(R, M) = \text{BLAKE-512}(R \| M)$; $\mathcal{F}_a(A, K) = \text{BLAKE-256}(K \| A)$; and $\mathcal{F}(M, K) = \text{BLAKE-512}(K \| M)$.

5 Fast Software Implementation

The fastest arithmetic units of most modern microprocessors are vector units. Instead of performing a certain arithmetic operation on scalar inputs, they perform the same operation in parallel on multiple values kept in vector registers. Not surprisingly, many speed records for cryptographic algorithms are held by implementations that make efficient use of these vector units. Also not surprisingly, many modern cryptographic primitives are designed with vectorizability in mind. In this section we describe how to efficiently implement SPHINCS-256 using vector instructions, more specifically the AVX2 vector instructions in Intel Haswell processors. All cycle counts reported in this section are measured on one core of an Intel Xeon E3-1275 CPU running at 3.5 GHz. We followed the standard practice of turning off Turbo Boost and hyperthreading for our benchmarks.

The AVX2 Instruction Set. The Advanced Vector Extensions (AVX) were introduced by Intel in 2011 with the Sandy Bridge microarchitecture. The extensions feature 16 vector registers of size 256 bits. In AVX, those registers can only be used as vectors of 8 single-precision floating-point values or vectors of 4 double-precision floating-point values. This functionality was extended in AVX2, introduced with the Haswell microarchitecture, to also support arithmetic on 256-bit vectors of integers of various sizes. We use these AVX2 instructions for 8-way parallel computations on 32-bit integers.

Vectorizing Hash Computations. The two low-level operations in SPHINCS that account for most of the computations are the fixed-input-size hash functions F and H . The SPHINCS-256 instantiation of F and H internally uses the ChaCha permutation. We now discuss vectorized computation of this permutation.

An obvious approach is to use the same parallelism exploited in [3, Section 3.1.3], which notes that the core operations in ChaCha and BLAKE “can be computed in four parallel branches”. Most high-speed implementations of BLAKE use this internal 4-way parallelism for vector computations.

However, it is much more efficient to vectorize across multiple independent computations of F or H. The most obvious reason is that the ChaCha permutation operates on 32-bit integers which means that 4-way-parallel computation can only make use of half of the 256-bit AVX vector registers. A second reason is that internal vectorization of ChaCha requires relatively frequent shuffling of values in vector registers. Those shuffles do not incur a serious performance penalty, but they are noticeable. A third reason is that vectorization is not the only way that modern microprocessors exploit parallelism. Instruction-level parallelism is used on pipelined processors to hide latencies and superscalar CPUs can even execute multiple independent instruction in the same cycle. A non-vectorized implementation of ChaCha has 4-way instruction-level parallelism which makes very good use of pipelining and superscalar execution. A vectorized implementation of ChaCha has almost no instruction-level parallelism and suffers from serious instruction-latency penalties.

Our 8-way parallel implementation of F takes 420 cycles to hash 8 independent 256-bit inputs to 8 256-bit outputs. Our 8-way parallel implementation of H takes 836 cycles to hash 8 independent 512-bit inputs to 8 256-bit outputs. These speeds assume that the inputs are interleaved in memory. Interleaving and de-interleaving data means transposing an 8×8 32-bit-word matrix.

This vectorization across 8 simultaneous hash computations is suitable for the two main components in the SPHINCS signature generation, namely HORST signing and WOTS authentication-path computations, as described below. The same approach also generalizes to other instantiations of F and H, although for some functions it is more natural to use 64-bit words.

HORST Signing. The first step in HORST signature generation is to expand the secret seed into a stream of $t \cdot n = 16\,777\,216$ bits (or 2 MB). This pseudo-random stream forms the 2^{16} secret keys of HORST. We use ChaCha12 for this seed expansion, more specifically Andrew Moon’s implementation of ChaCha12, which SUPERCOP identifies as the fastest implementation for Haswell CPUs. The seed expansion costs about 1 814 424 cycles.

The costly part of HORST signing is to first evaluate $F(\mathbf{sk}_i)$ for $i = 0, \dots, t - 1$, and then build the binary hash tree on top of the $F(\mathbf{sk}_i)$ and extract nodes which are required for the 32 authentication paths. SPHINCS-256 uses $t = 2^{16}$ so we need a total of 65 536 evaluations of F and 65 535 evaluations of H. A streamlined vectorized implementation treats the HORST binary tree up to level 13 (the level with 8 nodes) as 8 independent sub-trees and vectorizes computations across these sub-trees. Data needs to be interleaved only once at the beginning (the HORST secret keys \mathbf{sk}_i) and de-interleaved at the very end (the 8 nodes on level 13 of the HORST tree). All computations in between are streamlined 8-way parallel computations of F and H on interleaved data. The final tree hashing from level 13 to the HORST root at level 16 needs only

7 evaluations of H. This is a negligible cost, even when using a slower non-vectorized implementation of H.

Note that, strictly speaking, we do not have to interleave input at all; we can simply treat the 2 MB output of ChaCha12 as already interleaved random data. However, this complicates compatible non-vectorized or differently vectorized implementations on other platforms.

WOTS Authentication Paths. Computing a WOTS authentication path consists of 32 WOTS key generations, each followed by an L-Tree computation. This produces 32 WOTS public keys, which form the leaves of a binary tree. Computing this binary tree and extracting the nodes required for the authentication path finishes this computation. The costly part of this operation is the computation of 32 WOTS public keys ($32 \cdot 15 \cdot 67 = 32\,160$ evaluations of F) and of 32 L-Tree computations ($32 \cdot 66 = 2\,112$ evaluations of H). For comparison, the binary tree at the end costs only 31 computations of H. Efficient vectorization parallelizes across 8 independent WOTS public-key computations with subsequent L-Tree computations. Data needs to be interleaved only once at the very beginning (the WOTS secret key) and de-interleaved once at the very end (the roots of the L-Trees). Again, all computations in between are streamlined 8-way parallel computations of F and H on interleaved data.

SPHINCS Signing Performance. Our software still uses some more transpositions of data than the almost perfectly streamlined implementation described above. With these transpositions and some additional overhead to xor hash inputs with masks, update pointers and indices etc., HORST signing takes 15 033 564 cycles. The lower bound from 65 536 evaluations of F and 65 535 evaluations of H is 10 289 047 cycles. The computation of one WOTS authentication path takes 2 586 784 cycles. The lower bound from 32 160 evaluations of F and 2 143 evaluations of H is 1 912 343 cycles. The complete SPHINCS-256 signing takes 51 636 372 cycles; almost all of these cycles are explained by one HORST signature and 12 WOTS authentication paths.

SPHINCS Key Generation and Verification. The by far most costly operation in SPHINCS is signing so we focused our optimization efforts on this operation. Some easily vectorizable parts of key generation and verification also use our high-speed 8-way vectorized implementations of F and H, but other parts still use relatively slow non-vectorized versions based on the ChaCha12 reference implementation in eBACS [10]. Our implementation of key generation takes 3 237 260 cycles. Our implementation of signature verification takes 1 451 004 cycles.

RAM Usage and Size. Our implementation is optimized for speed on large Intel processors where size and memory usage are typically only a minor concern. Consequently, we did not optimize for those parameters. For example, we keep the complete HORST tree in memory and then extract the hashes that are needed in the 32 authentication paths. This approach keeps the software simple, but if we wanted to save memory, we would instead use treehash [32] to construct the tree and extract and store required authentication-path entries on the fly.

Although the software is not optimized for memory usage, we do not need any dynamic memory allocations; all temporary data fits into the Linux default stack limit of 8 MB. The size of the complete signing software, including BLAKE for message hashing, is 104 KB.

Acknowledgments. Thanks to Christian Rechberger and Andrew Miller for helpful discussions on the topic.

References

1. Aumasson, J.-P., Bernstein, D.J.: SipHash: A Fast Short-Input PRF. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 489–508. Springer, Heidelberg (2012)
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)
3. Aumasson, J.-P., Henzen, L., Meier, W., Phan, R.C.-W.: SHA-3 proposal BLAKE. Submission to NIST (2008). <http://131002.net/blake/blake.pdf>
4. Aumasson, J.-P., Neves, S., Wilcox-O’Hearn, Z., Winnerlein, C.: BLAKE2: Simpler, Smaller, Fast as MD5. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 119–135. Springer, Heidelberg (2013)
5. Bernstein, D.J.: What output size resists collisions in a xor of independent expansions? ECRYPT Hash Workshop (2007)
6. Bernstein, D.J.: ChaCha, a variant of Salsa20. In: SASC 2008: The State of the Art of Stream Ciphers (2008)
7. Bernstein, D.J.: The Salsa20 Family of Stream Ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 84–97. Springer, Heidelberg (2008)
8. Bernstein, D.J.: Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? In: Workshop Record of SHARCS’09: Special-purpose Hardware for Attacking Cryptographic Systems (2009)
9. Bernstein, D.J.: Extending the Salsa20 nonce. In: Symmetric Key Encryption Workshop 2011 (2011)
10. Bernstein, D.J., Lange, T.: eBACS: ECRYPT benchmarking of cryptographic systems. <http://bench.cr.yp.to> (accessed May 25, 2014)
11. Bernstein, D.J., Lange, T.: Non-uniform Cracks in the Concrete: The Power of Free Precomputation. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 321–340. Springer, Heidelberg (2013)
12. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT Hash Workshop (2007)
13. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The road from Panama to Keccak via RadioGatún, Dagstuhl Seminar Proceedings (2009)
14. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: Keyak 1 (2014)
15. 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)

16. 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)
17. Buchmann, J., García, L.C.C., Dahmen, E., Döring, M., Klintsevich, E.: CMSS – An Improved Merkle Signature Scheme. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 349–363. Springer, Heidelberg (2006)
18. 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)
19. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice Signatures and Bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013)
20. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 336–354. Springer, Heidelberg (2012)
21. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. *Journal of Cryptology* **10**(3), 151–161 (1997)
22. 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)
23. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press, Cambridge (2004)
24. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing* **17**(2), 281–308 (1988)
25. Hülsing, A.: Practical Forward Secure Signatures using Minimal Security Assumptions. PhD thesis, TU Darmstadt (2013)
26. 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)
27. Hülsing, A., Rausch, L., Buchmann, J.: Optimal parameters for XMSS^{MT}. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CD-ARES Workshops 2013. LNCS, vol. 8128, pp. 194–208. Springer, Heidelberg (2013)
28. Kilian, J., Rogaway, P.: How to protect DES against exhaustive key search (an analysis of DESX). *Journal of Cryptology* **14**(1), 17–35 (2001)
29. Kurosawa, K.: Power of a public random permutation and its application to authenticated-encryption. *Cryptology ePrint Archive, Report 2002/127* (2002)
30. Lamport, L.: Constructing digital signatures from a one way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory (1979)
31. Langley, A.: TLS symmetric crypto (2014). <https://www.imperialviolet.org/2014/02/27/tlssymmetriccrypto.html>
32. Merkle, R.C.: A Certified Digital Signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
33. Pieprzyk, J., Wang, H., Xing, C.: Multiple-time signature schemes against adaptive chosen message attacks. In: Matsui, M., Zuccherato, R. (eds.) SAC 2003. LNCS 3006, pp. 88–100. Springer, Heidelberg (2004)
34. Reyzin, L., Reyzin, N.: Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In: Batten, L.M., Seberry, J. (eds.) ACISP 2002. LNCS, vol. 2384, pp. 144–153. Springer, Heidelberg (2002)

35. Song, F.: A Note on Quantum Security for Post-Quantum Cryptography. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 246–265. Springer, Heidelberg (2014)
36. Suzuki, K., Tonien, D., Kurosawa, K., Toyota, K.: Birthday Paradox for Multicollisions. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 29–40. Springer, Heidelberg (2006)

A Security Properties

In this appendix we give the basic definitions for security properties we use.

Existential Unforgeability Under Adaptive Chosen Message Attacks. The standard security notion for digital signature schemes is existential unforgeability under adaptive chosen message attacks (EU-CMA) [24] which is defined using the following experiment. By $\text{Dss}(1^n)$ we denote a signature scheme with security parameter n .

Experiment $\text{Exp}_{\text{Dss}(1^n)}^{\text{EU-CMA}}(\mathcal{A})$

$$\begin{aligned} (\text{sk}, \text{pk}) &\leftarrow \text{kg}(1^n) \\ (M^*, \sigma^*) &\leftarrow \mathcal{A}^{\text{sign}(\text{sk}, \cdot)}(\text{pk}) \\ \text{Let } \{(M_i, \sigma_i)\}_1^q \text{ be the query-answer pairs of } \text{sign}(\text{sk}, \cdot). \\ \text{Return 1 iff } \text{vf}(\text{pk}, M^*, \sigma^*) = 1 \text{ and } M^* \notin \{M_i\}_1^q. \end{aligned}$$

A signature scheme is called existentially unforgeable under a q adaptive chosen message attack if any PPT adversary making at most q queries, has only negligible success probability in winning the above game.

An EU-CMA secure one-time signature scheme (OTS) is a signature scheme that is existentially unforgeable under a 1-adaptively chosen message attack.

Hash Function Families. We now provide definitions of the security properties of hash function families that we use, namely one-wayness, second-preimage resistance, undetectability and pseudorandomness. In the following let $n \in \mathbb{N}$ be the security parameter, $m, k = \text{poly}(n)$, $\mathcal{H}_n = \{\text{H}_K : \{0, 1\}^m \rightarrow \{0, 1\}^n \mid K \in \{0, 1\}^k\}$ a family of functions. (In the description of SPHINCS we actually omit the key K in many cases for readability.)

We define the security properties in terms of the success probability of an adversary \mathcal{A} against the respective property. A function family \mathcal{H}_n is said to provide a property if the success probability of any probabilistic polynomial-time adversary against this property is negligible. We begin with the success probability of an adversary \mathcal{A} against the one-wayness (OW) of a function family \mathcal{H}_n .

$$\begin{aligned} \text{Succ}_{\mathcal{H}_n}^{\text{OW}}(\mathcal{A}) &= \Pr [K \xleftarrow{\$} \{0, 1\}^k; M \xleftarrow{\$} \{0, 1\}^m, Y \leftarrow \text{H}_K(M), \\ &\quad M' \leftarrow \mathcal{A}(K, Y) : Y = \text{H}_K(M')] . \end{aligned}$$

We next define the success probability of an adversary \mathcal{A} against second-preimage resistance (SPR).

$$\text{Succ}_{\mathcal{H}_n}^{\text{SPR}}(\mathcal{A}) = \Pr[\mathcal{K} \xleftarrow{\$} \{0,1\}^k; M \xleftarrow{\$} \{0,1\}^m, M' \leftarrow \mathcal{A}(K, M) : (M \neq M') \wedge (\text{H}_K(M) = \text{H}_K(M'))].$$

To define undetectability, assume the following two distributions over $\{0,1\}^n \times \{0,1\}^k$. A sample (U, K) from the first distribution $\mathcal{D}_{\text{UD}, \mathcal{U}}$ is obtained by sampling $U \xleftarrow{\$} \{0,1\}^n$ and $K \xleftarrow{\$} \{0,1\}^k$ uniformly at random from the respective domains. A sample (U, K) from the second distribution $\mathcal{D}_{\text{UD}, \mathcal{H}}$ is obtained by sampling $K \xleftarrow{\$} \{0,1\}^k$ and then evaluating H_K on a uniformly random bit string, i.e., $\mathcal{U}_m \xleftarrow{\$} \{0,1\}^m, U \leftarrow \text{H}_K(\mathcal{U}_m)$. The success probability of an adversary \mathcal{A} against the undetectability of \mathcal{H}_n is defined as:

$$\text{Succ}_{\mathcal{H}_n}^{\text{UD}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\mathcal{D}_{\text{UD}, \mathcal{U}}} = 1] - \Pr[\mathcal{A}^{\mathcal{D}_{\text{UD}, \mathcal{H}}} = 1]|,$$

where $\mathcal{A}^{\text{dist}}$ denotes that \mathcal{A} has oracle access to some oracle that outputs samples from distribution dist .

The fourth notion we use is pseudorandomness of a function family (PRF). In the definition of the success probability of an adversary against pseudorandomness the adversary gets black-box access to an oracle Box . Box is either initialized with a function from \mathcal{H}_n or a function from the set $\mathcal{G}(m, n)$ of all functions with domain $\{0,1\}^m$ and range $\{0,1\}^n$. The goal of the adversary is to distinguish both cases:

$$\text{Succ}_{\mathcal{H}_n}^{\text{PRF}}(\mathcal{A}) = \left| \Pr[\text{Box} \xleftarrow{\$} \mathcal{H}_n : \mathcal{A}^{\text{Box}(\cdot)} = 1] - \Pr[\text{Box} \xleftarrow{\$} \mathcal{G}(m, n) : \mathcal{A}^{\text{Box}(\cdot)} = 1] \right|.$$

Subset-Resilient Functions. We now recall the definition of subset resilience from [34]. Let $\mathcal{H} = \{\text{H}_{i,t,k}\}$ be a family of functions, where $\text{H}_{i,t,k}$ maps a bit string of arbitrary length to a subset of size at most k of the set $[t-1]$. (As for hash functions in the description of SPHINCS we omit the key and assume the used function is randomly selected from a family using the uniform distribution.) Moreover, assume that there is a polynomial-time algorithm that, given $i, 1^t, 1^k$ and M , computes $\text{H}_{i,t,k}(M)$. Then \mathcal{H} is called γ -subset resilient if the following success probability is negligible for any probabilistic polynomial-time adversary \mathcal{A} :

$$\begin{aligned} \text{Succ}_{\mathcal{H}}^{\gamma\text{-sr}}(\mathcal{A}) &= \Pr_i \left[(M_1, M_2, \dots, M_{\gamma+1}) \leftarrow \mathcal{A}(i, 1^t, 1^k) \right. \\ &\quad \left. \text{s.t. } \text{H}_{i,t,k}(M_{\gamma+1}) \subseteq \bigcup_{j=1}^{\gamma} \text{H}_{i,t,k}(M_j) \right] \end{aligned}$$

Evaluating Implementations

Making Masking Security Proofs Concrete

Or How to Evaluate the Security of Any Leaking Device

Alexandre Duc¹, Sebastian Faust^{1,2}, and François-Xavier Standaert³(✉)

¹ EPFL, Lausanne, Switzerland

{alexandre.duc, sebastian.faust}@epfl.ch

² Horst Görtz Institute, Ruhr-University Bochum, Bochum, Germany

³ ICTEAM/ELEN/Crypto Group, Université catholique de Louvain,
Louvain-la-Neuve, Belgium

fstandae@uclouvain.be

Abstract. We investigate the relationships between theoretical studies of leaking cryptographic devices and concrete security evaluations with standard side-channel attacks. Our contributions are in four parts. First, we connect the formal analysis of the masking countermeasure proposed by Duc et al. (Eurocrypt 2014) with the Eurocrypt 2009 evaluation framework for side-channel key recovery attacks. In particular, we re-state their main proof for the masking countermeasure based on a mutual information metric, which is frequently used in concrete physical security evaluations. Second, we discuss the tightness of the Eurocrypt 2014 bounds based on experimental case studies. This allows us to conjecture a simplified link between the mutual information metric and the success rate of a side-channel adversary, ignoring technical parameters and proof artifacts. Third, we introduce heuristic (yet well-motivated) tools for the evaluation of the masking countermeasure when its independent leakage assumption is not perfectly fulfilled, as it is frequently encountered in practice. Thanks to these tools, we argue that masking with non-independent leakages may provide improved security levels in certain scenarios. Eventually, we consider the tradeoff between measurement complexity and key enumeration in divide-and-conquer side-channel attacks, and show that it can be predicted based on the mutual information metric, by solving a non-linear integer programming problem for which efficient solutions exist. The combination of these observations enables significant reductions of the evaluation costs for certification bodies.

1 Introduction

Side-channel attacks are an important concern for the security of cryptographic hardware, and masking is one of the most investigated solutions to counteract them. Its underlying principle is to randomize any sensitive data manipulated by a leaking implementation by splitting it into d shares, and to perform all the

computations on these shared values only. Intuitively, such a process is expected to force the adversary to combine several leakages corresponding to the different shares in order to recover secret information. As a result, it has first been shown by Chari et al. that the measurement complexity of a specialized attack – namely a single-bit Differential Power Analysis (DPA) [35] – against a carefully implemented masked computation (i.e. where the leakages of all the shares are independent and sufficiently noisy) increases exponentially with d [14]. Following this seminal work, a number of progresses have been made in order to state the security guarantee of masking in both general and rigorous terms. For example, Ishai, Sahai and Wagner introduced a compiler (next referred to as the ISW compiler), able to encode any circuit into an equivalent (secret-shared) one, and proved its security against so-called probing adversaries, able to read a bounded number of wires in the implementation [33]. A practical counterpart to these results was published at Asiacrypt 2010, where Standaert et al. analyzed the security of several masked implementations [61], using the information theoretic framework introduced in [60]. While this analysis was specialized to a few concrete case studies, it allowed confirming the exponential security increase provided by masking against actual leakages, typically made of a noisy but arbitrary function of the target device’s state. Following, Faust et al. attempted to analyze the ISW compiler against more realistic leakage functions, and succeeded to prove its security against computationally bounded (yet still unrealistic) ones, e.g. in the AC^0 complexity class [25]. Prouff and Rivain then made a complementary step towards bridging the gap between the theory and practice of masking schemes, by providing a formal information theoretic analysis of a wide (and realistic) class of so-called noisy leakage functions [49]. Eventually, Duc et al. turned this analysis into a simulation-based security proof, under standard conditions (i.e. chosen-message rather than random-message attacks, without leak-free components, and with reduced noise requirements) [22]. The central and fundamental ingredient of this last work was a reduction from the noisy leakage model of Prouff and Rivain to the probing model of Ishai et al.

Our Contribution. In view of this state-of-the-art, one of the main remaining questions regarding the security of the masking countermeasure is whether its proofs can be helpful in the security evaluation of concrete devices. That is, can we state theorems for masking so that the hypotheses can be easily fulfilled by hardware designers, and the resulting guarantee is reflective of the actual security level of the target implementation. For this purpose, we first observe that the proofs in [22, 49] express their hypothesis for the amount of noise in the shares’ leakages based on a statistical distance. This is in contrast with the large body of published work where the mutual information metric introduced in [60] is estimated for various implementations (e.g. [4, 12, 27, 30, 32, 42, 50, 51, 57, 63, 66]). Since the latter metric generally carries more intuition (see, e.g. [3] in the context of linear cryptanalysis), and benefits from recent advances in leakage certification, allowing to make sure that its estimation is accurate and based on sound assumptions [23], we first provide a useful link between the statistical distance and mutual information, and also connect them with easy-to-interpret

(but more specialized) tools such as the Signal-to-Noise Ratio (SNR). We then re-state the theorems of Duc et al. based on the mutual information metric in two relevant scenarios. Namely, we consider both the security of an idealized implementation with a “leak-free refreshing” of the shares, and the one of a standard ISW-like encoding (i.e. capturing any type of leaking computation).

Interestingly, the implementation with leak-free refreshing corresponds to the frequently investigated (practical) context where a side-channel attack aims at key recovery, and only targets the d shares’ leakage of a so-called sensitive intermediate variable (i.e. that depends on the plaintext and key) [17]. So despite being less interesting from a theoretical point of view, this scenario allows us to compare the theorem bounds with concrete attacks. Taking advantage of this comparison, we discuss the bounds’ tightness and separate parameters that are physically motivated from more “technical ones” (that most likely result of proof artifacts). As a result, we conjecture a simplified link between the mutual information metric and the success rate of a side-channel adversary, which allows accurate approximations of the attacks’ measurement complexity at minimum (evaluation) cost. We further illustrate that the noise condition for masking has a simple and intuitive interpretation when stated in terms of SNR.

Next, we note that the published results about masking (including the previously mentioned theorems and conjecture) assume independence between the leakages corresponding to different shares in an implementation. Yet, concrete experiments have shown that small (or even large) deviations from this assumption frequently occur in practice (see, e.g. [5, 16, 41, 54]). Hence, we complete our discussion by providing sound heuristics to analyze the impact of “non-independent leakages” which allow, for the first time, to evaluate and predict the security level of a masked implementation in such imperfect conditions.

Eventually, we consider the tradeoff between measurement complexity and time complexity in the important context of divide-and-conquer attacks. Previously known approaches for this purpose were based on launching key enumeration and/or rank estimation algorithms for multiple attacks, and to average results to obtain a success rate [64, 65]. We provide an alternative solution, where success rates (possibly obtained from estimations of the mutual information metric) are estimated/bounded for all the target key bytes of the divide-and-conquer attack first, and the impact of enumeration is evaluated only once afterwards. We also connect the problem of approximating the enumeration cost for a given number of measurements with a non-linear integer programming problem, and provide simple heuristics to estimate bounds on this enumeration cost.

Summarizing, the combination of these observations highlights that the security evaluation of a masked implementation boils down to the estimation of the mutual information between its shares and their corresponding leakages. Incidentally, the tools introduced in this paper apply identically to unprotected implementations, or implementations protected with other countermeasures, as long as one can estimate the same mutual information metric for the target intermediate values. Therefore, our results clarify the long standing open question whether the (informal) link between information theoretic and security metrics

in the Eurocrypt 2009 evaluation framework [60] can be proved formally. They also have important consequences for certification bodies, since they translate the (worst-case) side-channel evaluation problem into the well-defined challenge of estimating a single metric, leading to significantly reduced evaluation costs.

Notations. We next use capital letters for random variables, small caps for their realizations and hats for estimations. Vectors will be denoted with bold notations, functions with sans serif fonts, and sets with calligraphic ones.

2 Background

2.1 Leakage Traces and Assumptions

Let y be a n -bit sensitive value manipulated by a leaking device. Typically, it could be the output of an S-box computation such that $y = \mathbf{S}(x \oplus k)$ with n -bit plaintext/key words x and k . Let y_1, y_2, \dots, y_d be the d shares representing y in a Boolean masking scheme (i.e. $y = y_1 \oplus y_2 \oplus \dots \oplus y_d$). In a side-channel attack, the adversary is provided with some information (aka leakage) on each share. Typically, this leakage takes the form of a random variable \mathbf{L}_{y_i} that is the output of a leakage function L with y_i and a noise variable \mathbf{R}_i as arguments:

$$\mathbf{L}_{y_i} = \mathsf{L}(y_i, \mathbf{R}_i). \quad (1)$$

The top of Fig. 1 represents a leakage trace corresponding to the manipulation of d shares. Concretely, each subtrace \mathbf{L}_{y_i} is a vector of which the elements represent time samples. Whenever accessing a single time sample t , we use the notation $\mathbf{L}_{y_i}^t = \mathsf{L}^t(y_i, \mathbf{R}_i^t)$. From this general setup, a number of assumptions are frequently used in the literature. We will consider the following three.

a. Selection of Points-of-Interest / Dimensionality Reduction. For convenience, a number of attacks start with a pre-processing in order to reduce each leakage subtrace \mathbf{L}_{y_i} to a scalar random variable L_{y_i} . Such a pre-processing is motivated both by popular side-channel distinguishers such as Correlation Power Analysis (CPA) [11], which can only deal with univariate data, and by the easier representation of small dimensional data spaces. In this respect, even distinguishers that naturally extend towards multivariate data (such as Template attacks (TA) [15], Linear Regression (LR) [58] or Mutual Information Analysis (MIA) [28]) generally benefit from some dimensionality reduction. This step can be achieved heuristically, by looking for leakage samples where one distinguisher works best, or more systematically using tools such as Principal Component Analysis (PCA) [2] or Linear Discriminant Analysis (LDA) [59]. An example of reduced leakage trace is represented at the bottom of Fig. 1.

b. Additive Noise. A standard assumption in the literature is to consider leakage functions made of a deterministic part $\mathsf{G}(y_i)$ and additive noise \mathbf{N}_i [40]:

$$\mathbf{L}_{y_i} = \mathbf{L}(y_i, \mathbf{R}_i) \approx \mathsf{G}(y_i) + \mathbf{N}_i. \quad (2)$$

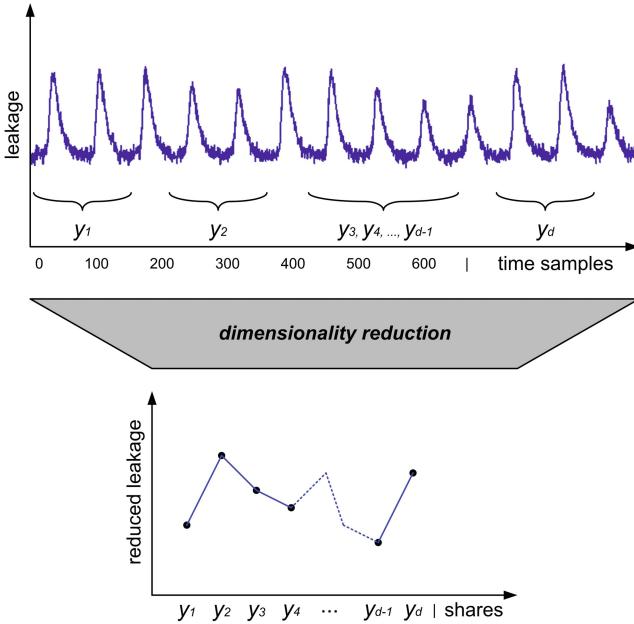


Fig. 1. Leakage trace & reduced leakage trace of a d -shared secret

For example, a typical setting is to assume reduced leakages to be approximately generated as the combination of a Hamming weight function (or some other simple function of the shares’ bits [58]) with additive Gaussian noise.

c. Independence Condition. A secure implementation of the masking countermeasure requires that the leakage vectors \mathbf{L}_{y_i} are independent random variables. If respected, it implies that a d -share masking will lead to a $(d-1)^{\text{th}}$ -order secure implementation as defined in [17]. That is, it guarantees that every d -tuple of leakage vectors is independent of any sensitive variable. This means that any adversary targeting the implementation will have to “combine” the information of at least d shares, and that extracting information from these d shares will require to estimate a d^{th} -order moment of the leakage PDF (conditioned on a sensitive variable) – a task that becomes exponentially hard in d if the noise is sufficient. As witnessed by several prior works, this condition may be hard to fulfill in practice. In software implementations, it typically requires avoiding transition-based leakages (i.e. leakages that depend on the distance between shares rather than directly on the shares) [5, 16]. In hardware implementations, physical defaults such as glitches are another usual issue that can invalidate the independence assumption [41], which motivates various research efforts to mitigate this risk, both at the hardware level (e.g. [43]) and at the algorithmic level (e.g. [46]).

Note that only this last (independence) assumption is strictly needed for the following proofs of Sect. 3 to hold. By contrast, the previous assumptions (a)

and (b) will be useful to provide practical intuition in Sect. 4. Furthermore, it is worth noting that slight deviations from this independence assumption (i.e. slight dependencies between the shares' leakages) may still lead to concrete security improvements, despite falling outside the proofs' formal guarantees. Such (practically meaningful) contexts will be further analyzed in Sect. 4.2.

2.2 Evaluation Metrics

Following [60], one generally considers two types of evaluation metrics for leaking cryptographic devices. First, information theoretic metrics aim to capture the amount of information available in a side-channel, independent of the adversary exploiting it. Second, security metrics aim to quantify how this information can be exploited by some concrete adversary. As will be clear next, the two types of metrics are related. For example, in the context of standard DPA attacks [41], they both measure the prediction of the (true) leakage function with some model, the latter usually expressed as an estimation of the leakage Probability Density Function (PDF). Yet they differ since information theoretic metrics only depend on the leakage function and model, while security metrics also depend on the adversary's computational power. For example, the capacity to enumerate key candidates may improve security metrics, but has no impact on information theoretic ones [64, 65]. Our goal in the following is to draw a formal connection between information theoretic and security metrics, i.e. between the amount of leakage provided by an implementation and its (worst-case) security level.

In the case of masking, proofs informally state that “*given that the leakage of each share is independent of each other and sufficiently noisy, the security of the implementation increases exponentially in the number of shares*”. So we need the two types of metrics to quantify the noise condition and security level.

b. Metrics to Quantify the Noise Condition. In general (i.e. without assumptions on the leakage distribution), the noise condition on the shares can be expressed with an information theoretic metric. The Mutual Information (MI) advocated in [60] is the most frequently used candidate for this purpose:

$$\text{MI}(Y_i; \mathbf{L}_{Y_i}) = H[Y_i] + \sum_{y_i \in \mathcal{Y}} \Pr[y_i] \cdot \sum_{\mathbf{l}_{y_i} \in \mathcal{L}} \Pr[\mathbf{l}_{y_i} | y_i] \cdot \log_2 \Pr[y_i | \mathbf{l}_{y_i}], \quad (3)$$

where we use the notation $\Pr[Y_i = y_i] =: \Pr[y_i]$ when clear from the context. Note that whenever trying to compute this quantity from an actual implementation, evaluators face the problem that the leakage PDF is unknown and can only be sampled and estimated. As a result, one then computes the Perceived Information (PI), which is the evaluator's best estimate of the MI [54]:

$$\hat{\text{PI}}(Y_i; \mathbf{L}_{Y_i}) = H[Y_i] + \sum_{y_i \in \mathcal{Y}} \Pr[y_i] \cdot \sum_{\mathbf{l}_{y_i} \in \mathcal{L}} \Pr_{\text{chip}}[\mathbf{l}_{y_i} | y_i] \cdot \log_2 \hat{\Pr}_{\text{model}}[y_i | \mathbf{l}_{y_i}], \quad (4)$$

with \Pr_{chip} the true chip distribution that can only be sampled and $\hat{\Pr}_{\text{model}}$ the adversary's estimated model. For simplicity, we will ignore this issue and use the MI in our discussions (conclusions would be identical with the PI).

Interestingly, when additionally considering reduced leakages with additive Gaussian noise, and restricting the evaluation to so-called “first-order information” (i.e. information lying in the first-order statistical moments of the leakage PDF, which is typically the case for the leakage of each share), simpler metrics can be considered [40]. For example, the SNR introduced by Mangard at CT-RSA 2004 in [38] is of particular interest for our following discussions:

$$\text{SNR} = \frac{\hat{\text{var}}_{Y_i}(\hat{E}_{n_i}(L_{Y_i}))}{\hat{E}_{Y_i}(\hat{\text{var}}_{n_i}(L_{Y_i}))}, \quad (5)$$

where \hat{E} is the sample mean operator and $\hat{\text{var}}$ is the sample variance. Summarizing, stating the noise condition based on the MI metric is more general (as it can capture any leakage PDF). By contrast, the SNR provides a simpler and more intuitive condition in a more specific but practically relevant context.

Eventually, the previous works of Prouff–Rivain and Duc et al. [22, 49] consider the following Statistical Distance (SD) to state their noise condition:

$$\text{SD}(Y_i; Y_i | \mathbf{L}_{Y_i}) = \sum_{\mathbf{l}_{y_i} \in \mathcal{L}} \Pr[\mathbf{l}_{y_i}] \cdot d(Y_i; Y_i | \mathbf{l}_{y_i}), \quad (6)$$

with d the Euclidean norm in [49] and $d(X_1, X_2) = \frac{1}{2} \sum_{x \in \mathcal{X}} |\Pr[X_1 = x] - \Pr[X_2 = x]|$ in [22]. In their terminology, a leakage function \mathbf{L} is then called “ δ -noisy” if $\delta = \text{SD}(Y_i; Y_i | \mathbf{L}_{Y_i})$, which was useful to connect different leakage models.

As previously mentioned, some of these metrics can be related under certain conditions. For example, in the context of univariate Gaussian random variables, the MI can be approximated from Pearson’s correlation coefficient [40], which was also connected to the SNR by Mangard [38]. The combination of those links corresponds to the classical MI bound in Cover and Thomas [19]:

$$\text{MI}(Y_i; \mathbf{L}_{Y_i}) \approx -\frac{1}{2} \log \left(1 - \left(\frac{1}{\sqrt{(1 + \frac{1}{\text{SNR}})}} \right)^2 \right) \leq \frac{1}{2} \log \left(1 + \text{SNR} \right). \quad (7)$$

In Sect. 3.1, we show that the MI and SD metrics can be connected as well.

c. Metrics to Quantify the Security Result. Quantifying security requires defining the adversary’s goal. Current side-channel attacks published in the literature mostly focus on key recovery. In this context, one can easily evaluate the exploitation of the leakages with the success rate defined in [60], i.e. the probability that an adversary recovers the key given the observation of some (typically known or chosen) plaintexts, ciphertexts and leakages. We will next denote it with SR^{kr} . Key recovery is a weak security notion from a cryptographic point of view. As a result, rigorous proofs for masking such as the one of Duc et al. in [22] rather define security using the standard real/ideal world paradigm, which consider two settings: the ideal world where the adversary attacks the algorithm of a cryptographic scheme in a black-box way, and the real world where he additionally obtains leakages. A scheme is said to be secure in the real world, if for any

adversary in the real world there exists an adversary in the ideal world. In other words: any attack that can be carried out given the leakages can also be carried out in a black-box manner. A proof of security usually involves constructing an efficient simulator that is able to simulate the leakages just giving black-box access to the attacked cryptographic scheme. Whenever considering this (standard) indistinguishability-based security notion, we will denote the adversary's success probability of distinguishing the two worlds with SR^{dist} .

3 Making Proofs Concrete: Theory

In this section, we discuss theoretical tweaks allowing to improve the concreteness of masking proofs. For this purpose, we recall three important leakage models that are relevant for our work. First, the t -probing and ϵ -probing (aka random probing) models were introduced in [33]. In the former one, the adversary obtains t intermediate values of the computation (e.g. can probe t wires if we compute in binary fields). In the latter one, he obtains each of these intermediate values with probability ϵ , and gets \perp with probability $1 - \epsilon$ (where \perp means no knowledge). Using a Chernoff-bound it is easy to show that security in the t -probing model reduces to security in the ϵ -probing model for certain values of ϵ . Second, the noisy leakage model describes many realistic side-channel attacks and allows an adversary to obtain each intermediate value perturbed with a δ -noisy leakage function L [49]. As mentioned in the previous section, a leakage function L is called δ -noisy if for a uniformly random variable Y (over the field \mathbb{F}) we have $\text{SD}(Y; Y|L_Y) \leq \delta$. In contrast with the conceptually simpler ϵ -probing model, the adversary obtains noisy leakages on each intermediate variable. For example, in the context of masking, he obtains $L(Y_i, R)$ for all the shares Y_i , which is more reflective of actual implementations where the adversary can potentially observe the leakage of all these shares, since they are all present in leakage traces such as in Fig. 1. Recently, Duc et al. showed that security against probing attacks implies security against noisy leakages (up to a factor $|\mathbb{F}|$, where \mathbb{F} is the underlying field in which the operations are carried out) [22]. In the rest of this section, we first connect the statistical distance SD with the mutual information metric MI, which shows that both can be used to quantify the noise condition required for masking. Next, we provide alternative forms for the theorems of Duc et al. and show (i) the security of the encoding used in (e.g. Boolean) masking and (ii) the security of a complete circuit based on the ISW compiler.

3.1 From Statistical Distance to MI

The results from Duc et al. require to have a bound on the SD between the shares and the shares given the leakage. For different reasons, expressing this distance based on the MI metric may be more convenient in practice (as witnessed by the numerous works where this metric has been computed, for various types of devices, countermeasures and technologies – see the list in introduction). For example, the MI metric is useful to determine whether the leakage model used

in a standard DPA is sound (see the discussion in Sect. 4.1) and for analyzing the impact of key enumeration in divide-and-conquer attacks (see the discussion in Sect. 4.3). Very concretely, Equations (3) and (4) are also expressed in a way that requires summing over the intermediate values first and on the leakages afterwards, which corresponds to the way security evaluations are performed (i.e. fix the target device’s state, and then perform measurements). Thus, we now show how to express the SD in function of the MI. We use a previous result from Dodis [21], which proofs follows [9] that we rephrase with our notations.

Lemma 1 ([21], Lemma 6). *Let Y_i and \mathbf{L}_{Y_i} be two random variables. Then:*

$$\frac{1}{2} \left(\sum_{(y \in \mathcal{Y}, \ell \in \mathcal{L})} |\Pr[Y_i = y, \mathbf{L}_{Y_i} = \ell] - \Pr[Y_i = y] \Pr[\mathbf{L}_{Y_i} = \ell]| \right)^2 \leq \text{MI}(Y_i; \mathbf{L}_{Y_i}).$$

Using this lemma, we can now express the SD in function of the MI as follows.

Theorem 1. *Let Y_i and \mathbf{L}_{Y_i} be two random variables. Then:*

$$2 \cdot \text{SD}(Y_i; Y_i | \mathbf{L}_{Y_i})^2 \leq \text{MI}(Y_i; \mathbf{L}_{Y_i}).$$

Proof. The proof follows the proof of [8], Lemma 4.4. We have:

$$\begin{aligned} & \sum_{(y \in \mathcal{Y}, \ell \in \mathcal{L})} |\Pr[Y_i = y, \mathbf{L}_{Y_i} = \ell] - \Pr[Y_i = y] \Pr[\mathbf{L}_{Y_i} = \ell]|, \\ &= \sum_{\ell \in \mathcal{L}} \Pr[\mathbf{L}_{Y_i} = \ell] \sum_{y \in \mathcal{Y}} |\Pr[Y_i = y | \mathbf{L}_{Y_i} = \ell] - \Pr[Y_i = y]|, \\ &= 2 \cdot \text{SD}(Y_i; Y_i | \mathbf{L}_{Y_i}). \end{aligned}$$

The final result directly derives from Lemma 1. □

3.2 Security of the Encoding

In this section, we analyze the security of an encoding when m measurements are performed and the encoding is refreshed between each measurements using a leak-free gate. More precisely, we assume that a secret y is secret-shared into d shares y_1, \dots, y_d , using an additive masking scheme over a finite field \mathbb{F} . Between each measurement, we assume that we take fresh y_1, \dots, y_d values such that $y = y_1 + \dots + y_d$ (e.g. it could be the Boolean encoding of Sect. 2.1). We also assume that this refreshing process does not leak and first recall a previous result from [22] that relates the random probing model to the noisy model. For conciseness, we call an adversary in the random-probing model a “random-probing adversary”, an adversary in the δ -noisy model a “ δ -noisy adversary”, and an adversary having access to leakages such that $\text{MI}(Y; Y | \mathbf{L}_Y) \leq \delta$ a “ δ -MI-adversary”. However, note that the physical noise (and its quantification with the MI) is a property of the implementation rather than of the adversary.

Lemma 2 ([22], Lemma 3). *Let \mathcal{A} be a δ -noisy adversary on \mathbb{F}^d . Then, there exists a $\delta \cdot |\mathbb{F}|$ -random-probing adversary \mathcal{S} on \mathbb{F}^d such that for every (y_1, \dots, y_d) , \mathcal{A} and \mathcal{S} produce the same view when applied on (y_1, \dots, y_d) .*

This result enables us to work directly in the random-probing model instead of the noisy leakage model. Next, we study the security of the encoding. As mentioned in introduction, the adversary's goal in this case is to recover the encoded value, which is equivalent to key recovery if this value is a key. In order to make it completely comparable with actual attacks, we also add the number of measurements m used by the adversary as a parameter in our bounds.

Theorem 2. *Let d be the number of shares used for a key encoding, m be the number of measurements, and $\text{MI}(Y_i; \mathbf{L}_{Y_i}) \leq t$ for some $t \leq 2/|\mathbb{F}|^2$. Then, if we refresh the encoding in a leak-free manner between each measurement, the probability of success of a key recovery adversary under independent leakage is:*

$$\text{SR}^{\text{kr}} \leq 1 - \left(1 - \left(|\mathbb{F}| \sqrt{t/2}\right)^d\right)^m. \quad (8)$$

Proof. In the random probing model with parameter ϵ , an adversary learns nothing about the secret if there is at least one share that did not leak. Since all the measurements are independent and we use leak-free refreshing gates, we have:

$$\text{SR}^{\text{kr}} \leq 1 - (1 - \epsilon^d)^m. \quad (9)$$

Let A be a t -MI-adversary on \mathbb{F}^d . From Theorem 1, we know that A implies a $\sqrt{t/2}$ -noisy-adversary on \mathbb{F}^d and, by Lemma 2, we obtain a $|\mathbb{F}| \sqrt{t/2}$ -random-probing adversary on \mathbb{F}^d . Letting $\epsilon := |\mathbb{F}| \sqrt{t/2}$ in (9) gives us the result. \square

Note that Equation (9) focuses on the impact of the adversary's measurement complexity m on the success rate, which is usually the dominating factor in concrete side-channel analyses. Yet, the impact of time complexity when considering key enumeration will be discussed in Sect. 4.3. Besides and for readability, this equation only includes the terms corresponding to attacks taking advantage of the leakages. We ignore the additional terms corresponding to mathematical cryptanalysis (e.g. exhaustive search) that should be added for completeness. In order to allow us comparing this result with the case where we study the security of a complete circuit encoded with the ISW compiler, we also write our result according to the following corollary (which is less general than Theorem 2).

Corollary 1. *Let d be the number of shares used for a key encoding and m the number of measurements. Then, if we refresh the encoding in leak-free manner between each measurement and for any $\alpha > 0$, the probability of success of a key recovery adversary under independent leakage is:*

$$\text{SR}^{\text{kr}} \leq m \cdot \exp(-\alpha d), \quad (10)$$

if we have:

$$\text{MI}(Y_i; \mathbf{L}_{Y_i}) \leq 2 \left(\frac{1}{e^\alpha |\mathbb{F}|} \right)^2. \quad (11)$$

Proof. We have:

$$1 - (1 - \epsilon^d)^m \leq m e^{\log(\epsilon)d}.$$

We want $\log(\epsilon) = -\alpha$. Hence, from Theorem 2, we get our result. \square

3.3 Security of the Whole Circuit

In this section, we restate the theorems from Duc et al. when securing a whole circuit with the seminal ISW compiler. The main theorem from [22] bounds the probability of success of a distinguishing adversary in the noisy leakage model. We provide an alternative version of their theorem and, as in the previous section, we relate it to the mutual information instead of the statistical distance.

Theorem 3. *Suppose that we have a circuit of size $|\Gamma|$ protected with the ISW compiler with d shares. Then, the probability of success of a distinguishing adversary under independent leakage is:*

$$\text{SR}^{\text{dist}} \leq |\Gamma| \cdot \exp\left(-\frac{d}{12}\right) = |\Gamma| \cdot 2^{\left(-\frac{d \cdot \log_2(e)}{12}\right)} \leq |\Gamma| \cdot 2^{-d/9}, \quad (12)$$

if we have:

$$\text{MI}(Y_i; \mathbf{L}_{Y_i}) \leq 2 \cdot \left(\frac{1}{|F| \cdot (28d + 16)}\right)^2. \quad (13)$$

Similarly to what we did in the previous section, we also write this corollary.

Corollary 2. *Suppose that we have a circuit of size $|\Gamma|$ protected with the ISW compiler with d shares. Then, if $\text{MI}(Y_i, \mathbf{L}_{Y_i}) \leq t$, a distinguisher adversary under independent leakage needs:*

$$d \geq \frac{1 - 16|F|\sqrt{\frac{1}{2}t}}{28|F|\sqrt{\frac{1}{2}t}} \quad (14)$$

shares in order to obtain:

$$\text{SR}^{\text{dist}} \leq |\Gamma| \cdot \exp\left(-\frac{d}{12}\right) \leq |\Gamma| \cdot \exp\left(-\frac{1 - 16|F|\sqrt{\frac{1}{2}t}}{336|F|\sqrt{\frac{1}{2}t}}\right). \quad (15)$$

Note that the ISW compiler can actually be used to efficiently compute any circuit. For example, the work of Rivain and Prouff at CHES 2010 showed how to adapt the compiler to $|F| = 256$ which leads to efficient masked implementations of the AES [56] (see also various following works such as [13, 18, 31, 57]).

4 Making Proofs Concrete: Practice

In this section, we complement the previous theoretical results with an experimental analysis. Our contributions are threefold. First, we provide and empirical evaluation of the encoding scheme in Sect. 3.2, which allows us to discuss the noise condition and tightness of the bounds in our proofs. We use this discussion to conjecture a simple connection between the mutual information metric and the success rate of a (worst-case) side-channel adversary, and argue that it can lead to quite accurate approximations of the attacks' measurement complexity. Next, we discuss possible deviations from the independent leakage assumption and provide tools allowing one to approximate the security level of concrete devices in such cases. Eventually, we consider the tradeoff between measurement complexity and time complexity in the context of divide-and-conquer side-channel attacks. We show how one can build a side-channel security graph (i.e. a plot of the adversary's success probability bounds in function of both parameters [65]), based only on the estimation of the MI metric for each share of a masking scheme. Along these lines, we eventually provide a formal justification for the physical security evaluation framework proposed at Eurocrypt 2009 [60].

4.1 Experimental Validation

In order to discuss the relevance of the proofs in the previous section, we take the (usual) context of standard DPA attacks defined in [40]. More precisely, we consider the simple case where an adversary targets a single S-box from a block cipher (e.g. the AES) as specified in Sect. 2.1, and obtains leakage variables $\mathbf{L}_{y_i} = \mathsf{L}(y_i, \mathbf{R}_i)$ for $1 \leq i \leq d$ (the case of multiple S-boxes will be studied in Sect. 4.3). For convenience, we mainly consider the context of mathematically-generated Gaussian Hamming weight leakages, where $\mathbf{L}_{y_i} = \mathsf{HW}(y_i) + N_i$, with HW the Hamming weight function and N_i a Gaussian-distributed noise, with variance σ^2 . In this respect, we note that we did not mount concrete attacks since we would have had to measure hundreds of different implementations to observe useful trends in practice. Our experiments indeed correspond to hundreds of different noise levels. Yet, we note that devices that exhibit close to Hamming weight leakages are frequently encountered in practice [39]. Furthermore, such a simulated setting is a well established tool to analyze masking schemes (see, e.g. [18] for polynomial masking, [4] for inner product masking and [12] for leakage squeezing). Besides, we also consider random Gaussian leakage functions, of which the deterministic part corresponds to random functions over \mathcal{Y} , to confirm that all the trends we put forward are also observed with leakage functions that radically differ from the usual Hamming weight one.

a. Computing the MI Metric. In this DPA setting, we aim to compute the MI between the key and the plaintext and leakages. For conciseness, we use the notations $\bar{Y} = [Y_1, \dots, Y_d]$ and $\bar{\mathbf{L}} = [\mathbf{L}_{Y_1}, \dots, \mathbf{L}_{Y_d}]$ for vectors containing the d shares and their corresponding leakages. Then we compute:

$$\begin{aligned} \text{MI}(K; X, \bar{\mathbf{L}}_Y) &= H[K] + \sum_{k \in \mathcal{K}} \Pr[k] \cdot \\ &\quad \sum_{x \in \mathcal{X}, \bar{y} \in \mathcal{Y}^d} \Pr[x, \bar{y}] \cdot \sum_{\bar{l}_y \in \mathcal{L}^d} \Pr[\bar{l}_y | k, x, \bar{y}] \cdot \log_2 \Pr[k | x, \bar{l}_y]. \end{aligned} \quad (16)$$

While this expression may look quite involved, we note that it is actually simple to estimate in practice, by sampling the target implementation. Evaluators just have to set keys k in their device and generate leakage traces corresponding to (known) plaintexts x and (unknown) shares \bar{y} . Say there are $|\mathcal{K}| = n_k$ key candidates and we generate n_t leakage traces \bar{l}_i , then, one just assigns probabilities \hat{p}_i^j to each key candidate k_j^* , for each measured trace, as in Table 1. This is typically done using TA or LR. Following, if the correct key candidate is k , the second line of (16) can be computed as $\hat{E}_i \log_2(\hat{p}_i^k)$. Note that whenever considering the standard DPA setting where the target operations follow a key addition, it is not even necessary to sum over the keys since $\text{MI}(K = k; X, \bar{\mathbf{L}}_Y)$ is identical for all k 's, thanks to the key equivalence property put forward in [40].

Table 1. Computing key candidate probabilities for MI metric estimation

	Key candidates			
State & leakage	k_1^*	k_2^*	\dots	$k_{n_k}^*$
$(k, x_1) \rightsquigarrow \bar{l}_1$	\hat{p}_1^1	\hat{p}_1^2	\dots	$\hat{p}_1^{n_k}$
$(k, x_2) \rightsquigarrow \bar{l}_2$	\hat{p}_2^1	\hat{p}_2^2	\dots	$\hat{p}_2^{n_k}$
\dots	\dots	\dots	\dots	\dots
$(k, x_{n_t}) \rightsquigarrow \bar{l}_{n_t}$	$\hat{p}_{n_t}^1$	$\hat{p}_{n_t}^2$	\dots	$\hat{p}_{n_t}^{n_k}$

Intuitively, $\text{MI}(K; X, \bar{\mathbf{L}}_Y)$ measures the amount of information leaked on the key variable K . The framework in [60] additionally defines a Mutual Information Matrix (MIM) that captures the correlation between any key k and key candidates k^* . Using our sampling notations, it can be simply defined as $\text{MIM}_{k,k^*} = H[K] + \sum_i \log_2(\hat{p}_i^{k^*})$, which directly leads to $\text{MI}(K; X, \bar{\mathbf{L}}_Y) = \hat{E}_k(\text{MIM}_{k,k})$.

b. Intuition Behind the Noise Condition. Theorems 2 and 3 both require that the MI between the shares and their corresponding leakage is sufficiently small. In other words, they require the noise to be sufficiently large. In this section, we compute the MI metric for both an unprotected implementation (i.e. $d = 1$) and a masked one (i.e. $d = 2$) in function of different parameters.¹ In order to illustrate the computation of this metric, we provide a simple open source code that evaluates the MI between a sensitive variable Y and its Hamming weights, for different noise levels, both via numerical integration (that is only possible for mathematically-generated leakages) and sampling (that is more reflective of the evaluation of an actual device) [1]. In the latter case, an evaluator additionally

¹ For the masked case, we consider univariate leakages corresponding to the parallel setting in [7], for which computing the MI is slightly faster than in the serial one.

has to make sure that his estimations are accurate enough. Tools for ensuring this condition are discussed in [23]. In the following, this sufficient sampling is informally confirmed by the smooth shape of our experimental curves.

We start with the simplest possible plot, where the MI metric is computed in function of the noise variance σ^2 . Figure 2 shows these quantities, both for Hamming weight leakage functions and for random ones with output range N_l (in the latter context, the functions for different N_l 's were randomly picked up prior to the experiments, and stable across experiments). We also considered different bit sizes ($n = 2, 4, 6, 8$). Positively, we see that in all cases, the curves reach a linear behavior, where the slope corresponds to the number of shares d . Since the independent leakage condition is fulfilled in these experiments, this d corresponds to the smallest key-dependent moment in the leakage distribution. And since the measurement (aka sampling) cost for estimating such moments is proportional to $(\sigma^2)^d$, we observe that the MI decreases exponentially in d for large enough noises. Note that this behavior is plotted for $d = 1, 2$, but was experimented for d 's up to 4 in [61], and in fact holds for any d , since it exactly corresponds to Theorem 2 in a context where its assumptions are fulfilled.

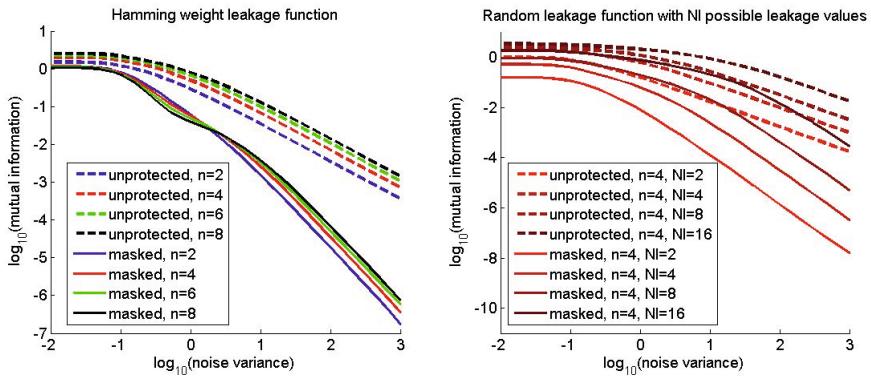


Fig. 2. MI metric in function of σ^2 . HW (left) and random (right) leakages.

Negatively, we also see that the noise level that can be considered as high enough depends on the leakage functions. For example, the random leakage functions in the right part of the figure have signals that vary from approximately $\frac{2}{4}$ for $N_l = 2$ to $\frac{16}{4}$ for $N_l = 16$. It implies that the linearly decreasing part of the curves is reached for larger noises in the latter case. Yet, this observation in fact nicely captures the intuition behind the noise condition. That is, the noise should be high enough for hiding the signal. Therefore, a very convenient way to express it is to plot the MI metric in function of shares' SNR, as in Fig. 3. Here, we clearly see that as soon as the SNR is below a certain constant (10^{-1} , typically), the shape of the MI curves gets close to linear. This corroborates the condition in Theorem 2 that masking requires $\text{MI}(K_i; X, \mathcal{L}_{Y_i})$ to be smaller than a given constant. Our experiments with different bit sizes also suggest that the $|\mathcal{F}|$

factor in this noise condition is a proof artifact. This is now formally proven by Dziembowski, Faust and Skorski in [24]. Of course, and as discussed in Sect. 2.2, the SNR metric is only applicable under certain conditions (univariate Gaussian leakages). So concretely, an evaluator may choose between computing it after dimensionality reduction (leading to a heuristic but intuitive condition), or to directly state the condition in function of the MI. For completeness, we also plot the MI metric for an unprotected and masked implementation in function of the share's MI in Appendix, Fig. 10. It clearly exhibits that as the share's MI decreases, this reduction is amplified by masking (exponentially in d).

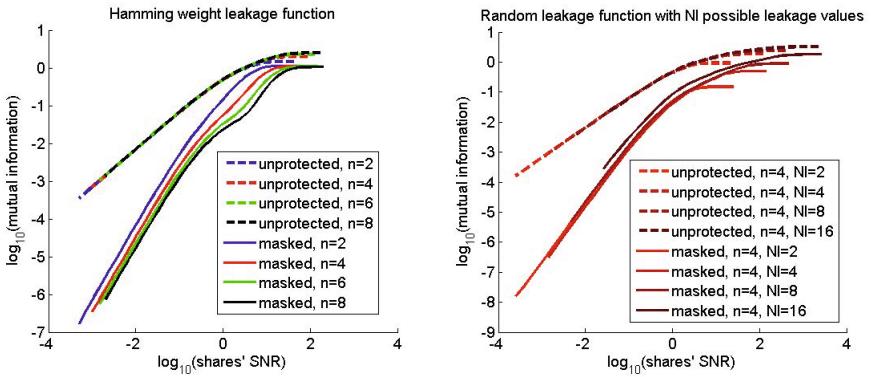


Fig. 3. MI metric in fct. of the shares' SNR. HW (left) and random (right) leakages.

c. Tightness of the Bounds. Given that the noise is high enough (as just discussed), Theorems 2 and 3 guarantee that the success rate of a side-channel adversary can be bounded based on the value of the share's leakage, measured with $\text{MI}(K_i; X, \mathbf{L}_{Y_i})$. This directly leads to useful bounds on the measurement complexity to reach a given success rate, e.g. from (8) we can compute:

$$m \geq \frac{\log(1 - \text{SR}^{\text{kr}})}{\log \left(1 - \left(|\mathbb{F}| \sqrt{\frac{\text{MI}(K_i; X, \mathbf{L}_{Y_i})}{2}} \right)^d \right)}. \quad (17)$$

We now want to investigate how tight this bound is. For this purpose, we compared it with the measurement complexity of concrete key recovery TA (using a perfect leakage model).² As previously mentioned, the $|\mathbb{F}|$ factor in this equation can be seen as a proof artifact related to the reduction in our theorems – so we tested a bound excluding this factor. For similar reasons, we also tested a bound additionally excluding the square root loss in the reductions (coming

² Our attacks exploit the leakages of an S-box output, as specified in Sect. 2.1. We took the PRESENT S-box for $n = 4$, the AES one for $n = 8$, and picked up two random S-boxes for $n = 2, 6$, as we did for the random leakage functions.

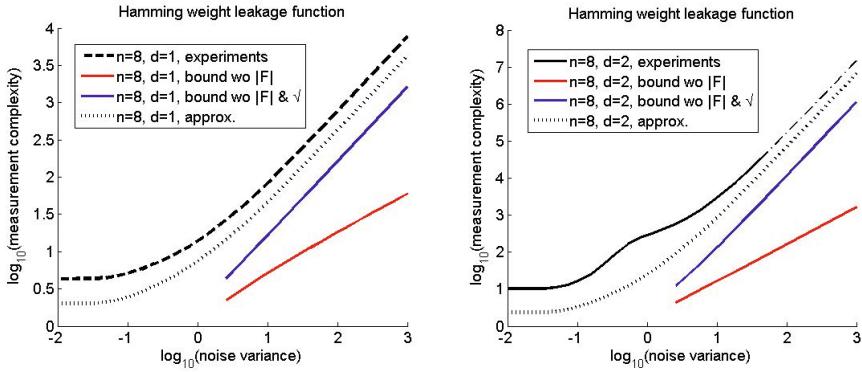


Fig. 4. Measurement complexity and bounds/approximations for concrete TA

from Theorem 1). As illustrated in Fig. 4, the measurement complexity of the attacks is indeed bounded by Equation (17), and removing the square root loss allows the experimental and theoretical curves to have similar slopes. The latter observation fits with the upper bound $\text{MI}(Y_i; \mathbf{L}_{Y_i}) \leq \frac{|\mathbb{F}|}{\ln(2)} \cdot \text{SD}(Y_i; Y_i \mid \mathbf{L}_{Y_i})$ given in [49] that becomes tight as the noise increases.³ As expected, the bounds become meaningless for too low noise levels (or too large SNRs, see Appendix, Fig. 11). Intuitively, this is because we reach success rates that are stuck to one when we deviate from this condition. For completeness, we added approximations obtained by normalizing the shares' MI by $H[K]$ to the figure, which provide hints about the behavior of a leaking device when the noise is too low.

Interestingly, these results also allow us to reach a comprehensive view of the parameters in Theorem 3, where the security of a complete circuit encoded according to the ISW compiler is proven. That is, in this case as well we expect the $|\mathbb{F}|$ and $1/9$ factors in Equation (12) to be due to proof technicalities. By contrast, the $|\Gamma|$ factor is physically motivated, since it corresponds to the size of the circuit and fits the intuition that more computations inevitably means more exploitable leakage. The d factor appearing in the noise condition of Equation (13) can also be explained, since it directly relates to the fact that in the ISW compiler, any multiplication will require to manipulate each share d times. It typically reflects the distance between standard (divide-and-conquer) side-channel attacks (such as analyzed in this section) and more powerful (multivariate) adversaries trying to exploit the leakage of all the intermediate computations in a block cipher, e.g. based on algebraic cryptanalysis (see [52, 53] and follow up works). Taking all these observations into account, we summarize the concrete security of any masking scheme with the following informal conjecture.

Informal Conjecture. *Suppose that we have a circuit of size $|\Gamma|$ masked with d shares such that the information leakage on each of these shares (using all available time samples) is bounded by $\text{MI}(Y_i; \mathbf{L}_{Y_i})$. Then, the probability of success of*

³ Since their inequality comes from a $\log(1+x) < \log(x)$ inequality that gets close to an equality when x gets close to 0, which happens for large noise levels.

a distinguishing adversary using m measurements and targeting a single element (e.g. gate) of the circuit under independent and sufficiently noisy leakage is:

$$\text{SR}_1^{\text{dist}} \leq 1 - (1 - \text{MI}(Y_i; \mathbf{L}_{Y_i})^d)^m , \quad (18)$$

and the probability of success targeting all $|\Gamma|$ elements independently equals:

$$\text{SR}_{|\Gamma|}^{\text{dist}} \leq 1 - (1 - \text{SR}_1^{\text{dist}})^{|\Gamma|} . \quad (19)$$

Interestingly, Equation (19) (like Theorem 3) assumes that the leakages of the $|\Gamma|$ gates (or target intermediate values) are exploited independently. This perfectly corresponds to the probing model in which the adversary gains either full knowledge or no knowledge of such computing elements. Thanks to [22], it also implies a similar result against noisy leakages if the noise condition is fulfilled. However, as the noise level decreases, some advanced (e.g. algebraic) side-channel attacks can sometimes take advantage of different computations jointly in a more efficient manner. Note that this informal conjecture is backed up by the results in [3] (Theorem 6) where a similar bound is given in the context of statistical cryptanalysis. By using the approximation $\log(1 - x) \approx -x$ that holds for x 's close to 0, Equation (18) directly leads to the following simple approximation of a standard DPA's measurement complexity for large noise levels:

$$m \geq \frac{\log(1 - \text{SR}_1^{\text{dist}})}{\log(1 - \text{MI}(Y_i; \mathbf{L}_{Y_i})^d)} \approx \frac{c}{\text{MI}(Y_i; \mathbf{L}_{Y_i})^d} , \quad (20)$$

where c is a small constant that depends on the target success rate. A similar approximation can be obtained from Equation (19) for multi-target attacks.

d. Relation with the Eurocrypt 2009 Evaluation Framework. The evaluation of leaking cryptographic implementations with a combination of information and security metrics was put forward by Standaert et al. at Eurocrypt 2009. In this reference, the authors showed a qualitative connection between both metrics. Namely, they proved that the model (i.e. the approximation of the leakage PDF) used by a side-channel adversary is sound (i.e. allows key recoveries) if and only if the mutual information matrix (defined in paragraph (a) of this section) is such that its diagonal values are maximum for each line. By contrast, they left the quantitative connection between these metrics as an open problem (i.e. does more MI imply less security?). Our results provide a formal foundation for this quantitative connection. They prove that for any implementation, decreasing the MI of the target intermediate values is beneficial to security. This can be achieved by ad hoc countermeasures, in which case it is the goal of an evaluation laboratory to quantify the MI metric, or by masking, in which case we can bound security based only on the value of this metric for each share taken separately.

4.2 Beyond Independent Leakage

The previous section evaluated an experimental setting where the leakage of each share is independent of each other, i.e. $\mathbf{L}_{y_i} = \mathbf{G}(y_i) + N_i$. But as discussed in

introduction, this condition frequently turns out to be hard to fulfill and so far, there are only limited (in)formal tools allowing to analyze the deviations from independent leakages that may be observed in practice. In order to contribute to this topic, we first launched another set of experiments (for 2-share masking), where the leakage of each share can be written as:

$$\begin{aligned} \mathbf{L}_{y_1} &= G_1(y_1) + f \cdot G_{1,2}(y_1, y_2) + N_1, \\ \mathbf{L}_{y_2} &= G_2(y_2) + f \cdot G_{2,1}(y_1, y_2) + N_2. \end{aligned}$$

Here the G_i functions manipulate the shares independently, while the $G_{i,j}$ functions depend on both shares. We additionally used the f (for flaw) parameter in order to specify how strongly we deviate from the independent leakage assumption. As in the previous section, we considered Hamming weight and random functions for all G 's (and we used $G_{i,j}(y_i, y_j) = G(y_i \oplus y_j)$ for illustration). Exemplary results of an information theoretic analysis in this context are given in Fig. 5 for the $n = 4$ -, and 8-bit cases (and in Appendix, Fig. 12 for the $n = 2$ - and 6-bit S-box cases). We mainly observe that as the noise increases, even small flaws are exploitable by an adversary. Indeed, breaking the independence condition makes smaller-order moments of the leakage distribution key-dependent. Consequently, for large enough noise, it is always this smaller-order moment that will be the most informative. This is empirically confirmed by the slopes of the IT curves in the figures, that gradually reach one rather than two.

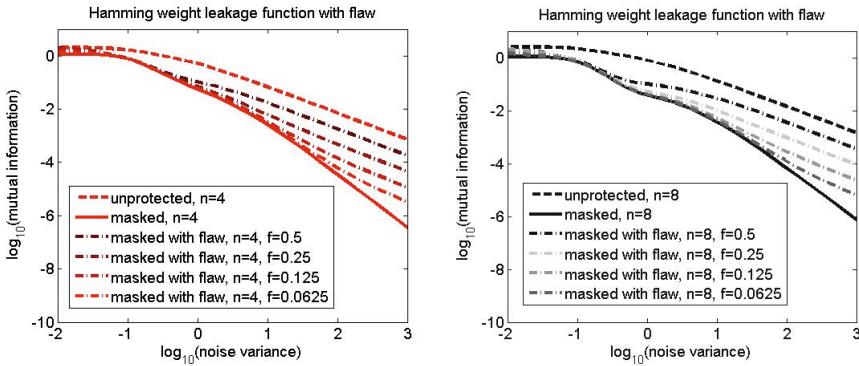


Fig. 5. MI metric for masked implementation with flaw ($n = 4, 8$)

Following these experiments, let us consider a chip that concretely exhibits such a flaw for a given noise level σ_{exp}^2 (corresponding to its actual measurements). Despite falling outside the masking proofs' guarantees, an important question is whether we can still (approximatively) predict its security level based on sound statistical tools. In this respect, a useful observation is that the MI metric cannot directly answer the question since it captures the information lying in all the statistical moments of the leakage PDF. So we need another ingredient

in order to reveal the informativeness of each moment of the leakage PDF, separately. The Moments-Correlating DPA (MC-DPA) recently introduced in [44] is a natural candidate for this purpose. We now describe how it can be used to (informally) analyze the security of a flawed masked implementation.

In this context, we first need to launch MC-DPA for different statistical moments, e.g. the first- and second-order ones in our 2-share example. They are illustrated by the circle and square markers in the left part of Fig. 6. For concreteness, we take the (most revealing) case where the second-order moment is more informative than the first-order one. Assuming that the noise condition in our theorems is fulfilled, the impact of increasing the noise on the value of the MC-DPA distinguisher can be predicted as indicated by the curves of the figure. That is, with a slope of $1/2$ for the first-order moment and a slope of 1 for the second-order one.⁴ Hence, we can directly predict the noise level $\sigma_{\text{exp}}^2 + \Delta$ such that the first-order moment becomes more informative. Eventually, we just observe that concrete side-channel attacks always exploit the smallest key-dependent moment in priority (which motivates the definition of the security-order for masking schemes [17]). So starting from the value of the MI at σ_{exp}^2 (represented by a circle in the right part of the figure), we can extrapolate that this MI will decrease following a curve with slope 2 until $\sigma_{\text{exp}}^2 + \Delta$ and a curve with slope 1 afterwards. Taking advantage of the theorems in the previous sections, this directly leads to approximations of the best attacks' measurement complexity. Furthermore, extending this reasoning to more shares and higher-order statistical moments is straightforward: it just requires to add MC-DPA curves in the left part of Fig. 6, and to always consider the one leading to the highest MC-DPA value to set the slope of the MI curves, in the right part of the figure. To the best of our knowledge, such figures (despite informal) provide the first concrete tools to approximate the security level in such contexts.

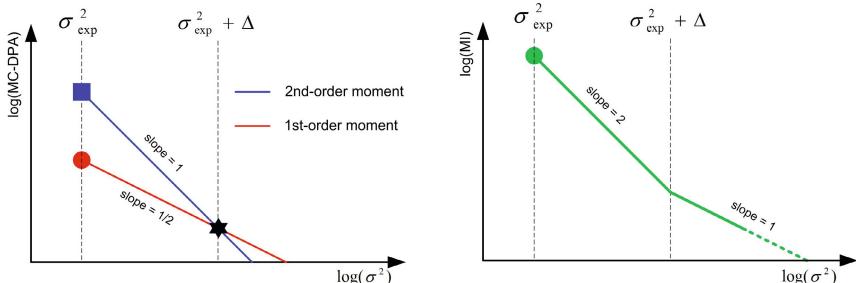


Fig. 6. Evaluating non-independent leakages with MC-DPA (left) and MI (right)

⁴ Slopes are divided by 2 when considering Pearson's correlation rather than the MI since this correlation is essentially proportional to the square root of the SNR. This is also reflected by the measurement complexity of CPA, that is proportional to the inverse of the squared correlation vs. the inverse of the MI for TA [62].

Note finally that the shape of the non-independent leakages (i.e. the $G_{i,j}$ functions) observed in practice highly depends on the implementations. For example in hardware, multiple shares can leak jointly in a hardly predictable manner [41, 54]. By contrast in software, the most usual issue (due to transition-based leakages) is easier to analyse [5]. It typically divides the order of the smallest key-dependent moment in the leakage distribution by two, which corresponds to the additional square root loss in the security bounds of Duc et al. when considering leakages that depend on two wires simultaneously (see [22], Sect. 5.5).

4.3 Exploiting Computational Power

In this section, we finally tackle the problem of divide-and-conquer DPA attacks, where the adversary aims to combine side-channel information gathered from a number of measurements, and computational power. That is, how to deal with the practically critical situation where the number of measurements available is not sufficient to exactly recover the key? As discussed in [64, 65], optimal enumeration and key ranking algorithms provide a concrete answer to this question. They allow building security graphs, where the success rate is plotted in function of a number of measurements and computing power, by repeating attacks multiple times. We next discuss more efficient and analytical strategies.

a. Why MI Is Not Enough? Whenever trying to exploit both side-channel leakage and brute-force computation (e.g. key enumeration) the most challenging aspect of the problem is to capture how measurements and computation actually combine. This is easily illustrated with the following example. Imagine two hypothetical side-channel attacks that both succeed with probability 1/100. In the first case, the adversary gains nothing with probability 99/100 and the full key with probability 1/100. In the second case, he always gains a set of 100 equally likely keys. Clearly, enumeration will be pretty useless in the first case, while extremely powerful in the second one. More generally, such examples essentially suggest that the computational cost of an enumeration does not only depend on the informativeness of the leakage function (e.g. measured with the MI) but also on its shape. For illustration, a line of the mutual information matrix computed from Hamming weight leakages for two noise levels is given in Fig. 7, where we can clearly identify the patterns due to this leakage model. While $\text{MIM}_{k,k}$ only corresponds to a single value of the matrix line (here $k = 111$), which bounds the measurement complexity to recover this key without additional computation (as previously discussed), how helpful is enumeration will additionally depend on the relative distance between the $\text{MIM}_{k,k}$ and MIM_{k,k^*} values [68]. Incidentally, this example also puts forward some limitations of the probing leakage model when measuring computational cost, since it describes an all-or-nothing strategy – as already mentioned in Sect. 4.1, paragraph (c) – which is not the case for the noisy leakage setting. Hence, whereas the probing model is easier to manipulate in proofs, and therefore useful to obtain asymptotic results, noisy leakages are a more accurate tool to quantify concrete security levels as in this section.

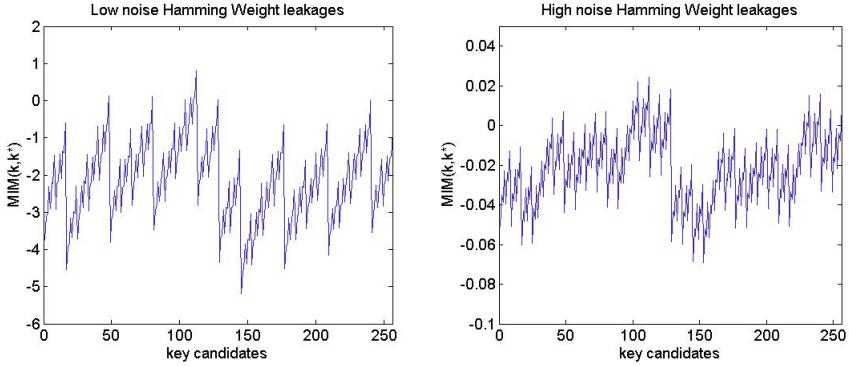


Fig. 7. Exemplary line of the mutual information matrix (for $k = 111$)

b. Measurement and Computational Bounds Per S-Box. Interestingly, one can easily derive bounds for attacks combining side-channel measurements and enumeration power against a single S-box, by re-using exactly the same material as we anyway need to estimate $\text{MI}(K; X, \mathbf{L}_{Y_i})$ for a single secret share. In the following, we will assume that the key equivalence property mentioned in Sect. 4.1, paragraph (a) holds, and focus on a single line of the mutual information matrix (if it does not, evaluators simply have to compute all its lines), next denoted as $\text{MIM}_{k,-}$. In order to characterize the distance between a key k and its close candidates k^* , we first sort this line and produce $s = \text{sort}(\text{MIM}_{k,-})$. As a result, the key candidate $k_{s(1)}^*$ is the best rated one (i.e. the correct k if the leakage model is sound), $k_{s(2)}^*$ is the second best, ... From there, we compute a “computational version” of the mutual information matrix as:

$$\text{MIM}_{k,k}^c = H[K_i] + E_j \left(\log \left(\sum_{l=1}^c \hat{p}_j^{s(l)} \right) \right). \quad (21)$$

It essentially corresponds to the amount of information an adversary obtains about a random variable that aggregates the c most likely key candidates. Assuming that these c key candidates are equally likely (which can only be pessimistic), it directly provides simple bounds on the success rate of an attack combining m measurements with the enumeration of c keys:

$$\text{SR}^{\text{kr}}(m, c) \leq 1 - (1 - (\text{MIM}_{k,k}^c)^d)^m, \quad (22)$$

For illustration, a couple of such bounds are given in Fig. 8, where we see the impact of increasing the number of shares d and number of measurements m . Note that despite requiring similar characterization efforts, these bounds are conceptually different from the previous approaches to approximate the success rate of side-channel attacks. In particular, works like [20, 26, 37, 55] are specific to popular distinguishers (and usually require specialized assumptions about the distribution of these distinguishers), while our results directly connect to security

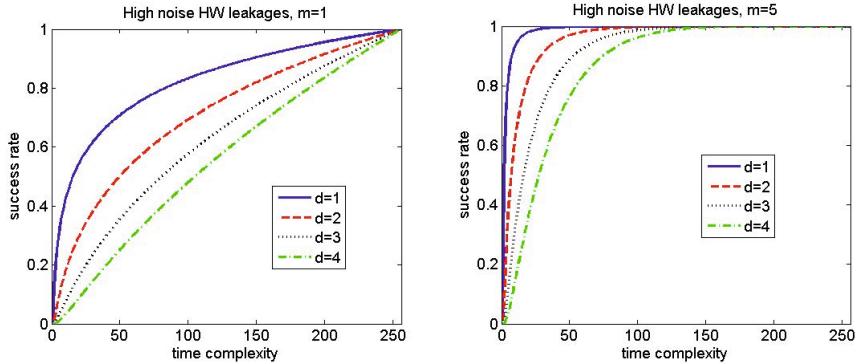


Fig. 8. Key recovery success rate against a single S-box, in function of the time complexity for the leakage function of Fig. 7 (right), after m measurements

proofs that are independent of the adversarial strategy and hold for any leakage distribution. Nevertheless, the only requirement to analyze the combination of multiple S-boxes in the next paragraph (c) is to have success rates curves for each S-box. So while this paragraph (b) describes an efficient way to build such curves, the following contribution is in fact general, and could be used as a complement to any security evaluation obtained for separate S-boxes.

c. Combining Multiple S-Boxes. We finally generalize our analysis of the previous paragraph to the case where we target n_s S-boxes (e.g. $n_s = 16$ for the AES), gained information about their respective input key bytes, and want to recover the full master key. We assume that we perform the same amount of measurements m on each S-box. This can be easily justified in practice, since a leakage trace usually contains samples corresponding to all S-boxes. By contrast, we make no assumption about how informative the leakages of each S-box are. For example, it could completely happen that one S-box is very leaky, and another one perfectly protected (so that enumeration is the only option to recover its corresponding key byte). As just explained, we then characterize the measurement vs. complexity tradeoff with n_s success rate curves $\text{SR}_i^{\text{kr}}(m, c_i)$ with $1 \leq i \leq n_s$. Typically, we will then set a limit β to the adversary's computational power and try to solve the following optimization problem:

$$\begin{aligned} & \max_{c_1, \dots, c_{n_s}} \quad \prod_{i=1}^{n_s} \text{SR}_i^{\text{kr}}(m, c_i) , \\ & \text{subject to} \quad \prod_{i=1}^{n_s} c_i \leq \beta . \end{aligned} \tag{23}$$

Taking the logarithm of both products, we get:

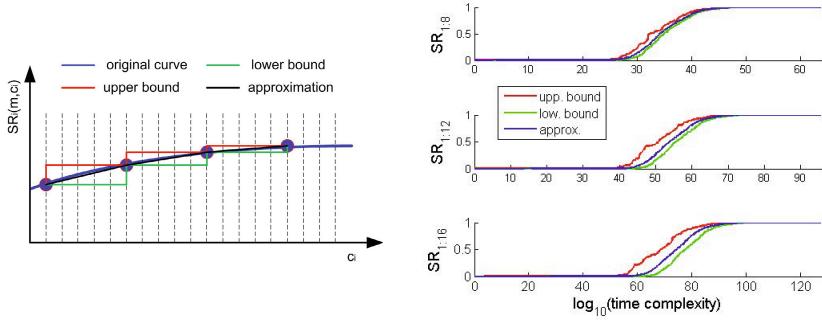


Fig. 9. Downsampling SR curves (left) and enumeration complexity bounds (right)

$$\begin{aligned} & \max_{c_1, \dots, c_{n_s}} \quad \sum_{i=1}^{n_s} \log \left(\text{SR}_i^{\text{kr}}(m, c_i) \right), \\ & \text{subject to} \quad \sum_{i=1}^{n_s} \log(c_i) \leq \log(\beta). \end{aligned} \quad (24)$$

For general functions SR_i^{kr} , this problem is known as a “separable, non-linear integer programming problem”. Surveys about non-linear integer programming problems are various (e.g. [10, 36]). There exist many well-studied heuristics to solve them, including branch-and-bounds and convex envelop techniques. Note that the problem generally becomes easier when dealing with convex functions.

We conclude this section with a simple and cheap heuristic algorithm which approximates well the optimal solution for the problem sizes and leakage functions we considered. The approach we propose is inspired by [29], and based on a tradeoff between the computational cost and accuracy of the solutions found, that is controlled by downsampling the success rate curves and keeping track of quantization errors. Intuitively, enumerating the combination of the possible success rates for two n -bit S-boxes requires the computation of 2^{2n} product complexities $c_i \cdot c_j$. Since combining more S-boxes exhaustively will increase the complexity exponentially (i.e. $2^{n_s \cdot n}$ for n_s n -bit S-boxes), the idea of our heuristic is simply to ignore some samples. Namely, we will fix a bound N_{\max} which will designate the maximum number of samples we save per success rate curve (or combination of them). Such a well-known downsampling process is informally illustrated in the left part of Fig. 9, where we can see that the original curve can be easily upperbounded and lowerbounded, since it is increasing.

This solution is described more formally in Algorithm 1 and works as follows. First, we downsample each success rate curve $\text{SR}_i(m, c)$ to N_{\max} linearly spaced points that we can write as N_{\max} pairs $(s_{i,1}, c_{i,1}), \dots, (s_{i,N_{\max}}, c_{i,N_{\max}})$. Next, we take the first S-Box and combine it with the second one, obtaining N_{\max}^2 values. These values are then downsampled again to N_{\max} linearly spaced points, so that we can iteratively combine them with the next S-boxes. We denote the aggregation of the i first success rate curves with $\text{SR}_{1:i}$. We also add an additional

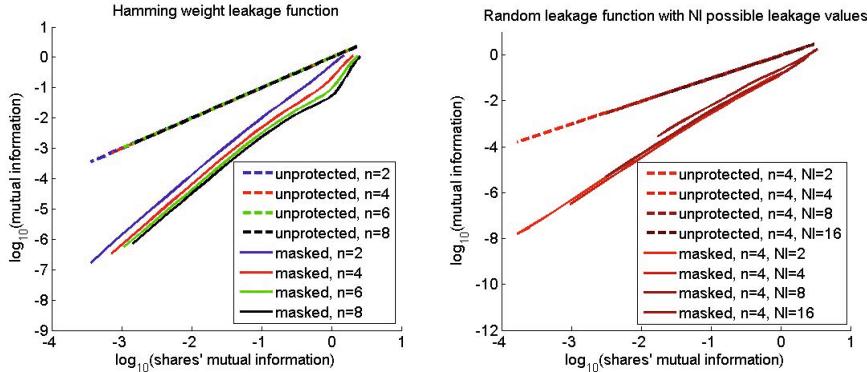


Fig. 10. MI metric in fct. of the shares' MI. HW (left) and random (right) leakages.

output to our algorithm, namely a list of complexities ℓ_i , describing how the effort is distributed among the S-boxes. Indeed, suppose for example that we combine the success rate pair $(0.1, 2^4)$ of S-box 1 with the success rate pair $(0.2, 2^5)$ of S-box 2. We obtain a success rate of 0.02 for a complexity of 2^9 , but nothing tells us how the effort is distributed between S-box 1 and S-box 2. Hence, we write the result as $(0.02, 2^9, \{2^4, 2^5\})$ which shows how the complexities are shared.

For illustration, the right part of Fig. 9 provides such bounds for the combination of 8, 12 and 16 AES S-boxes, for a noise level and number of measurement

Algorithm 1. Heuristic to combine the SR curves of n_s S-boxes

Require: Pairs $[(s_{i,1}, c_{i,1}), \dots, (s_{i,N_{\max}}, c_{i,N_{\max}})] =: \text{SR}_i$ for each S-box i .
Require: A bound N_{\max} on the number of samples and a bound β on the complexity.
Ensure: Triplets $(s_1, c_1, \ell_1), \dots, (s_{N_{\max}}, c_{N_{\max}}, \ell_{N_{\max}})$ approximating the success rate curve of the combination of the n_s S-Boxes, where the ℓ_i -s are ordered lists of complexities showing how they should be distributed among the S-boxes.

- 1: $\text{SR}_{1:1} \leftarrow [(s_{1,1}, c_{1,1}, \{c_{1,1}\}), \dots, (s_{1,N_{\max}}, c_{1,N_{\max}}, \{c_{1,N_{\max}}\})]$;
- 2: **for** $i = 2$ to n_s **do**
- 3: $\text{SR}_{1:i} \leftarrow \emptyset$;
- 4: ▷ Combination of aggregated S-Boxes $1 : i-1$ with S-Box i .
- 5: **for** $(s_j, c_j, \ell_j) \in \text{SR}_{1:i-1}$ **do** ▷ For all N_{\max} values in aggregated curve.
- 6: **for** $(s_{i,k}, c_{i,k}) \in \text{SR}_i$ **do** ▷ For all N_{\max} values of the new S-Box.
- 7: **if** $c_j \cdot c_{i,k} < \beta$ **then** ▷ If we did not reach the bound β .
- 8: $\text{SR}_{1:i} \leftarrow \text{SR}_{1:i} \cup (s_j \cdot s_{i,k}, c_j \cdot c_{i,k}, \ell_j \cup c_{i,k})$; ▷ Merge.
- 9: **end if**
- 10: **end for**
- 11: **end for**
- 12: ▷ Downsampling.
- 13: Sort $\text{SR}_{1:i}$ and keep only N_{\max} linearly spaced pairs.
- 14: **end for**
- 15: **return** $\text{SR}_{1:n_s}$

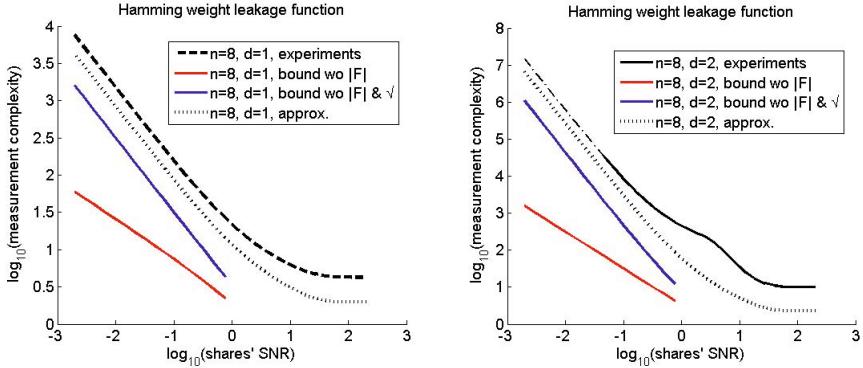


Fig. 11. Measurement complexity and bounds/approximations for concrete TA

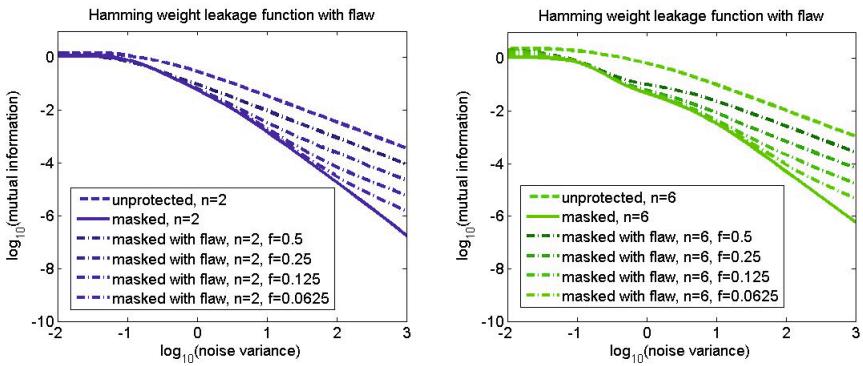


Fig. 12. MI metric for masked implementation with flaw ($n = 2, 6$)

such that the rank estimation problem is challenging (i.e. with full key rank for the 16-byte master key around 2^{80}). The complexity of this heuristic is proportional to $n_s \cdot (N_{\max}^2 + \log(N_{\max}))$ and the results in the figure were obtained within seconds of computation on a desktop computer, using a simple Matlab prototype code. We leave the investigation of better solutions to obtain accurate time complexity bounds with minimum efforts as a scope for further research.

Summarizing, our results show that the (complex) task of evaluating the worst-case security level of a masked implementation against (divide-and-conquer) DPA can be simplified to the evaluation of a couple of MI values, even in contexts where the independence assumption is not fulfilled. This provides a solid foundation for the Eurocrypt 2009 evaluation framework. It also makes it easier to implement, since success rate curves for full keys can now be derived from the MI values, rather than sampled experimentally by repeating (many) subkey recovery experiments and key rank estimations, which is an expensive

task. Taking advantage of the tools in this paper therefore allow reducing both the number of measurements and the time needed to evaluate leaking devices.

Acknowledgments. Alexandre Duc is supported by the Swiss National Science Foundation, grant 200021 143899/1. Sebastian Faust received funding from the Marie Curie IEF/FP7 project GAPS (grant 626467). François-Xavier Standaert is a research associate of the Belgian Fund for Scientific Research. This work has been funded in parts by the ERC project 280141 (CRASH).

References

1. <http://perso.uclouvain.be/fstandae/PUBLIS/154.zip>
2. Archambeau, C., Peeters, E., Standaert, F.-X., Quisquater, J.-J.: Template attacks in principal subspaces. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 1–14. Springer, Heidelberg (2006)
3. Baignères, T., Junod, P., Vaudenay, S.: How far can we go beyond linear cryptanalysis? In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 432–450. Springer, Heidelberg (2004)
4. Balasch, J., Faust, S., Gierlich, B., Verbauwhede, I.: Theory and practice of a leakage resilient masking scheme. In: Wang and Sako [67], pp. 758–775
5. Balasch, J., Gierlich, B., Gross, V., Reparaz, O., Standaert, F.-X.: On the Cost of Lazy Engineering for Masked Software Implementations. IACR Cryptology ePrint Archive 2014:413 (2014)
6. Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731. Springer, Heidelberg (2014)
7. Belaïd, S., Gross, V., Standaert, F.-X.: Masking and Leakage-Resilient Primitives: One, the Other(s) or Both? Cryptography and Communications **7**(1), 163–184 (2015)
8. Bellare, M., Tessaro, S., Vardy, A.: A Cryptographic Treatment of the Wiretap Channel. IACR Cryptology ePrint Archive 2012:15 (2012)
9. Bellare, M., Tessaro, S., Vardy, A.: Semantic security for the wiretap channel. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 294–311. Springer, Heidelberg (2012)
10. Bertsekas, D.P.: Nonlinear Programming. Athena Scientific (1999)
11. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
12. Carlet, C., Danger, J.-L., Guilley, S., Maghrebi, H.: Leakage Squeezing: Optimal Implementation and Security Evaluation. J. Mathematical Cryptology **8**(3), 249–295 (2014)
13. Carlet, C., Goubin, L., Prouff, E., Quisquater, M., Rivain, M.: Higher-order masking schemes for S-boxes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 366–384. Springer, Heidelberg (2012)
14. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener [69], pp. 398–412
15. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, C.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)

16. Coron, J.-S., Giraud, C., Prouff, E., Renner, S., Rivain, M., Vadnala, P.K.: Conversion of security proofs from one leakage model to another: A new issue. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 69–81. Springer, Heidelberg (2012)
17. Coron, J.-S., Prouff, E., Rivain, M.: Side channel cryptanalysis of a higher order masking scheme. In: Paillier, P., Verbauwheide, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 28–44. Springer, Heidelberg (2007)
18. Coron, J.-S., Prouff, E., Rivain, M., Roche, T.: Higher-order side channel security and mask refreshing. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 410–424. Springer, Heidelberg (2014)
19. Cover, T.M., Thomas, J.A.: Elements of Information Theory 2 edn. Wiley (2006)
20. Ding, A.A., Zhang, L., Fei, Y., Luo, P.: A statistical model for higher order DPA on masked devices. In: Batina and Robshaw [6], pp. 147–169
21. Dodis, Y.: Shannon impossibility, revisited. In: Smith, A. (ed.) ICITS 2012. LNCS, vol. 7412, pp. 100–110. Springer, Heidelberg (2012)
22. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: Nguyen and Oswald [45], pp. 423–440
23. Durvaux, F., Standaert, F.-X., Veyrat-Charvillon, N.: How to Certify the Leakage of a Chip? In: Nguyen and Oswald [45], pp. 459–476
24. Dziembowski, S., Faust, S., Skorski, M.: Noisy leakage revisited. In: The Proceedings of EUROCRYPT (to appear 2015)
25. Faust, S., Rabin, T., Reyzin, L., Tromer, E., Vaikuntanathan, V.: Protecting circuits from leakage: The computationally-bounded and noisy cases. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 135–156. Springer, Heidelberg (2010)
26. Fei, Y., Luo, Q., Ding, A.A.: A statistical model for DPA with novel algorithmic confusion analysis. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 233–250. Springer, Heidelberg (2012)
27. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine masking against higher-order side channel analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 262–280. Springer, Heidelberg (2011)
28. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald and Rohatgi [47], pp. 426–442
29. Glowacz, C., Gross, V., Poussier, R., Schueth, J., Standaert, F.-X.: Simpler and more efficient rank estimation for side-channel security assessment. In: IACR Cryptology ePrint Archive 2014:920 (2014)
30. Goubin, L., Martinelli, A.: Protecting AES with Shamir’s secret sharing scheme. In: Preneel and Takagi [48], pp. 79–94
31. Gross, V., Prouff, E., Standaert, F.-X.: Efficient masked s-boxes processing – A step forward –. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT. LNCS, vol. 8469, pp. 251–266. Springer, Heidelberg (2014)
32. Gross, V., Standaert, F.-X., Prouff, E.: Low entropy masking schemes, revisited. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 33–43. Springer, Heidelberg (2014)
33. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
34. Johansson, T., Nguyen, P.Q. (eds.): EUROCRYPT 2013. LNCS, vol. 7881. Springer, Heidelberg (2013)
35. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener [69], pp. 388–397

36. Li, D., Sun, X.: Nonlinear knapsack problems. In: Nonlinear Integer Programming. International Series in Operations Research & Management Science, vol. 84, pp. 149–207. Springer, US (2006)
37. Lomné, V., Prouff, E., Rivain, M., Roche, T., Thillard, A.: How to estimate the success rate of higher-order side-channel attacks. In: Batina and Robshaw [6], pp. 35–54
38. Mangard, S.: Hardware countermeasures against DPA – A statistical analysis of their effectiveness. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 222–235. Springer, Heidelberg (2004)
39. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks - Revealing the Secrets of Smart Cards. Springer (2007)
40. Mangard, S., Oswald, E., Standaert, F.-X.: One for All - All for One: Unifying Standard Differential Power Analysis Attacks. IET Information Security **5**(2), 100–110 (2011)
41. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked CMOS gates. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 351–365. Springer, Heidelberg (2005)
42. Medwed, M., Standaert, F.-X.: Extractors against Side-Channel Attacks: Weak or Strong? J. Cryptographic Engineering **1**(3), 231–241 (2011)
43. Moradi, A., Mischke, O.: Glitch-Free implementation of masking in modern FPGAs. In: HOST, pp. 89–95. IEEE (2012)
44. Moradi, A., Standaert, F.-X.: Moments-correlating DPA. In: IACR Cryptology ePrint Archive, 2014:409 (2014)
45. Nguyen, P.Q., Oswald, E. (eds.): EUROCRYPT 2014. LNCS, vol. 8441. Springer, Heidelberg (2014)
46. Nikova, S., Rijmen, V., Schläffer, M.: Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. J. Cryptology **24**(2), 292–321 (2011)
47. Oswald, E., Rohatgi, P. (eds.): CHES 2008. LNCS, vol. 5154. Springer, Heidelberg (2008)
48. Preneel, B., Takagi, T. (eds.): CHES 2011. LNCS, vol. 6917. Springer, Heidelberg (2011)
49. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Johansson and Nguyen [34], pp. 142–159
50. Prouff, E., Roche, T.: Attack on a higher-order masking of the AES based on homographic functions. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 262–281. Springer, Heidelberg (2010)
51. Renauld, M., Kamel, D., Standaert, F.-X., Flandre, D.: Information theoretic and security analysis of a 65-Nanometer DDSLL AES S-box. In: Preneel and Takagi [48], pp. 223–239
52. Renauld, M., Standaert, F.-X.: Algebraic side-channel attacks. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 393–410. Springer, Heidelberg (2010)
53. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N.: Algebraic side-channel attacks on the AES: Why time also matters in DPA. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 97–111. Springer, Heidelberg (2009)
54. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N., Kamel, D., Flandre, D.: A formal study of power variability issues and side-channel attacks for nanoscale devices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 109–128. Springer, Heidelberg (2011)

55. Rivain, M.: On the exact success rate of side channel analysis in the Gaussian model. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 165–183. Springer, Heidelberg (2009)
56. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
57. Roche, T., Prouff, E.: Higher-order Glitch Free Implementation of the AES using Secure Multi-Party Computation Protocols - Extended Version. *J. Cryptographic Engineering* **2**(2), 111–127 (2012)
58. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005)
59. Standaert, F.-X., Archambeau, C.: Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In: Oswald and Rohatgi [47], pp. 411–425
60. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
61. Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: Another look on second-order DPA. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 112–129. Springer, Heidelberg (2010)
62. Standaert, F.-X., Peeters, E., Rouvroy, G., Quisquater, J.-J.: An Overview of Power Analysis Attacks against Field Programmable Gate Arrays. *Proceedings of the IEEE* **94**(2), 383–394 (2006)
63. Standaert, F.-X., Petit, C., Veyrat-Charvillon, N.: Masking with randomized look up tables - Towards preventing side-channel attacks of all orders. In: Naccache, D. (ed.) Cryptography and Security: From Theory to Applications. LNCS, vol. 6805, pp. 283–299. Springer, Heidelberg (2012)
64. Veyrat-Charvillon, N., Gérard, B., Renaud, M., Standaert, F.-X.: An optimal key enumeration algorithm and its application to side-channel attacks. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 390–406. Springer, Heidelberg (2013)
65. Veyrat-Charvillon, N., Gérard, B., Standaert, F.-X.: Security evaluations beyond computing power. In: Johansson and Nguyen [34], pp. 126–141
66. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.-X.: Shuffling against side-channel attacks: A comprehensive study with cautionary note. In: Wang and Sako [67], pp. 740–757
67. Wang, X., Sako, K. (eds.): ASIACRYPT 2012. LNCS, vol. 7658. Springer, Heidelberg (2012)
68. Whitnall, C., Oswald, E.: A comprehensive evaluation of mutual information analysis using a fair evaluation framework. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 316–334. Springer, Heidelberg (2011)
69. Wiener, M. (ed.): CRYPTO 1999. LNCS, vol. 1666. Springer, Heidelberg (1999)

Ciphers for MPC and FHE

Martin R. Albrecht¹(✉), Christian Rechberger², Thomas Schneider³,
Tyge Tiessen², and Michael Zohner³

¹ Information Security Group, Royal Holloway, University of London, London, UK
martinralbrecht@googlemail.com

² Technical University of Denmark, Copenhagen, Denmark
{crec,tyti}@dtu.dk

³ TU Darmstadt, Darmstadt, Germany
{thomas.schneider,michael.zohner}@ec-spride.de

Abstract. Designing an efficient cipher was always a delicate balance between linear and non-linear operations. This goes back to the design of DES, and in fact all the way back to the seminal work of Shannon.

Here we focus, for the first time, on an extreme corner of the design space and initiate a study of symmetric-key primitives that minimize the multiplicative size and depth of their descriptions. This is motivated by recent progress in practical instantiations of secure multi-party computation (MPC), fully homomorphic encryption (FHE), and zero-knowledge proofs (ZK) where linear computations are, compared to non-linear operations, essentially “free”.

We focus on the case of a block cipher, and propose the family of block ciphers “LowMC”, beating all existing proposals with respect to these metrics by far. We sketch several applications for such ciphers and give implementation comparisons suggesting that when encrypting larger amounts of data the new design strategy translates into improvements in computation and communication complexity by up to a factor of 5 compared to AES-128, which incidentally is one of the most competitive classical designs. Furthermore, we identify cases where “free XORs” can no longer be regarded as such but represent a bottleneck, hence refuting this commonly held belief with a practical example.

Keywords: Block cipher · Multiplicative complexity · Multiplicative depth · Secure multiparty computation · Fully homomorphic encryption

1 Introduction

Modern cryptography developed many techniques that go well beyond solving traditional confidentiality and authenticity problems in two-party communication. Secure multi-party computation (MPC), zero-knowledge proofs (ZK) and fully homomorphic encryption (FHE) are some of the most striking examples.

In recent years, especially the area of secure multi-party computation has moved from a science that largely concerned itself with the mere existence of solutions towards considerations of a more practical nature, such as costs of

actual implementations for proposed protocols in terms of computational time, memory, and communication.

Despite important progress and existing proof-of-concept implementations, e.g. [MNPS04, PSSW09, HEKM11, NNOB12, KSS12, FN13, SS13], there exists a *huge cost gap* between employing cryptographic primitives in a traditional way and using them in the more versatile MPC context. As an example, consider implementations of the AES block cipher, a global standard for the bulk encryption of data. Modern processors achieve a single execution of the block cipher within a few hundred clock cycles (or even less than 100 clock cycles using AES-NI). However, realizing the same cipher execution in the context of an MPC protocol takes many billions of clock cycles and high communication volumes between the participating parties, e.g. several hundreds of Megabytes for two-party AES with security against malicious adversaries [PSSW09, NNOB12, KSS12, FN13, SS13, DZ13, LOS14, DLT14].

While our design approach is not specific to block ciphers but can be equally applied to e.g. hash functions, in this work, we propose block ciphers that are specifically designed for application in MPC and similar contexts. Traditionally, ciphers are built from linear and non-linear building blocks. These two have roughly similar costs in hardware and software implementations. In CMOS hardware, the smallest linear gate (XOR) is about 2-3 times larger than the smallest non-linear gate (typically, NAND). When implemented in an MPC protocol or a homomorphic encryption scheme, however, the situation is radically different: linear operations come almost for free, since they only incur local computation (resp. do not increase the noise much), whereas the bottleneck are non-linear operations that involve symmetric cryptographic operations and communication between parties (resp. increase the noise considerably). Our motivation hence comes from implementations of ciphers in the context of MPC, ZK, or FHE schemes where linear parts are much cheaper than non-linear parts.

This cost metric suggests a new way of designing a cipher where most of the cryptographically relevant work would be performed as linear operations and the use of non-linear operations is minimized. This design philosophy is related to the fundamental theoretical question of the minimal multiplicative complexity (MC) [BPP00] of certain tasks. Such extreme trade-offs were not studied before, as all earlier designs – due to their target platforms – faired better with obtaining a balance between linear and non-linear operations.

In this work we propose to start studying symmetric cryptography primitives with low multiplicative complexity in earnest. Earlier tender steps in this direction [GGNPS13, PRC12, GLSV14] were aimed at good cost and performance when implemented with side-channel attack countermeasures, and are not extreme enough for our purpose. Our question hence is: what is the minimum number of multiplications for building a secure block cipher? We limit ourselves to multiplications in GF(2) and motivate this as follows:

- By using Boolean circuits we decouple the underlying protocol / primitive (MPC protocol / ZK protocol / FHE scheme) from that of the cipher. Hence, the same cipher can be used for multiple applications.

- GF(2) is a natural choice for MPC protocols based on Yao or GMW (in the semi-honest setting, but also for their extensions to stronger adversaries), ZK protocols, as well as for fully or somewhat homomorphic encryption schemes (cf. Section 2 for details).

By nature of the problem, we are interested in two different metrics. One metric refers to what is commonly called multiplicative complexity (MC), which is simply the number of multiplications (AND gates) in a circuit, see e.g. [BPP00]. The second metric refers to the multiplicative depth of the circuit, which we will subsequently call ANDdepth. We note that already in [DSES14] it was observed that using ciphers with low ANDdepth is of central importance for efficient evaluations within homomorphic encryption schemes. Therefore, the authors of [DSES14] suggest to study block cipher designs that are optimized for low ANDdepth, a task to which we provide a first answer. Our work is somehow orthogonal to Applebaum et. al [AIK06], where the question of what can in principle be achieved in cryptography with shallow circuits was addressed.

This all motivates the following guiding hypothesis which we will test in this paper: “When implemented in practice, a block cipher design with lower MC and lower ANDdepth will result in lower executing times”. We note that the relatively low execution times often reported in the literature are *amortized* times, i.e. averaged over many calls of a cipher (in parallel). This, however, neglects the often important *latency*. Hence, another design goal in this work is to reduce this latency.

Outline and Contribution. In Section 2 we describe several schemes with “free XORs”. Then, in Section 3, we focus on an extreme corner of the design space of block ciphers and propose a new block-cipher design strategy that minimizes the multiplicative size and depth of the circuit describing it, beating all existing candidates by far with respect to these metrics. In terms of ANDdepth, the closest competitor is PRINCE. In terms of MC, the closest competitor turns out to be Simon. We give a high-level overview over a larger field of competing designs in Section 4. We analyse the security of our constructions in Section 5 and provide experimental evidence for the soundness of our approach in Section 6. In particular, our implementations outperform previously reported results in the literature, often by more than a factor 5 in MPC and FHE implementation settings. They also indicate that in the design space we consider, “free XORs” can no longer be regarded as free but significantly contribute to the overall cost, hence refuting this commonly held belief with a practical example. Finally, we describe our optimisation strategies for implementing our designs in the MPC and FHE case, which might be of independent interest.

Main Features and Advantages of LowMC

- Low ANDdepth, and low MC, which positively impacts the latency and throughput of the FHE, MPC, or ZK evaluation of the cipher.

- Partial Sbox layer.
- Security arguments against large classes of statistical attacks like differential attacks, similar to other state-of-the-art designs are given in Section 5. Zorro [GGNPS13] is the first SPN cipher in the literature that uses a non-full Sbox layer and is related to LowMC in this respect. However, recent attacks on Zorro that exploit this particular property [WWGY13, RASA14, GNPW13, BOOD+14], highlight the need to be very careful with this design strategy. In our analysis of LowMC in Section 5 we are able to take these into account.
- In contrast to other constructions, it is easy to obtain tight bounds on the MC and ANDdepth.
- The design is very flexible and allows for a unified description regardless of the blocksize.
- We explicitly de-couple the security claim of a block cipher from the block size.

2 Schemes

In this section we list several schemes for MPC, FHE, and ZK that benefit from evaluating our cipher. We give a list of example applications for LowMC in the full version of the paper.

2.1 Multi-Party Computation (MPC)

There are two classes of practically efficient secure multi-party computation (MPC) protocols for securely evaluating Boolean circuits where XOR gates are considerably cheaper (no communication and less computation) than AND gates.

The first class of MPC protocols has a constant number of rounds and their total amount of communication depends on the MC of the circuit (each AND gate requires communication). Examples are protocols based on Yao’s garbled circuits [Yao86] with the free XOR technique [KS08]. To achieve security against stronger (i.e., malicious or covert) adversaries, garbled circuit-based protocols apply the cut-and-choose technique where multiple garbled circuits are evaluated, e.g., [LP07, AL07, LPS08, PSSW09, LP11, SS11, KSS12, FN13, Lin13, HKE13, SS13, FJN14, HKK+14, LR14]; also MiniLEGO [FJN+13] falls into this class.

The second class of MPC protocols has a round complexity that is linear in the ANDdepth of the evaluated circuit (each AND gate requires interaction) and hence the performance depends on both, the MC and ANDdepth of the circuit. Examples are the semi-honest secure version of the GMW protocol [GMW87] implemented in [CHK+12, SZ13], and tiny-OT [NNOB12] with security against malicious adversaries.

2.2 Fully Homomorphic Encryption (FHE)

In all somewhat and fully homomorphic encryption schemes known so far XOR (addition) gates are considerably cheaper than AND (multiplication) gates. Moreover, XOR gates do not increase the noise much, whereas AND gates increase the noise considerably (cf. [HS14]). Hence, as in somewhat homomorphic encryption schemes the parameters must be chosen such that the noise of the result is low enough to permit decryption, the overall complexity depends on the ANDdepth.

2.3 Zero-Knowledge Proof of Knowledge (ZK)

In several zero-knowledge proof protocols XOR relations can be proven for free and the complexity essentially depends on the number of AND gates of the relation to be proven. Examples for such protocols are [BC86, BDP00] and the recently proposed highly efficient protocol of [JKO13] that requires only one evaluation of a garbled circuit [Yao86] and can make use of the free XOR technique [KS08].

3 Description of LowMC

LowMC is a flexible block cipher based on an SPN structure where the block size n , the key size k , the number of Sboxes m in the substitution layer and the allowed data complexity d of attacks can independently be chosen¹. The number of rounds needed to reach the security claims is then derived from these parameters.

To reduce the MC, the number of Sboxes applied in parallel can be reduced, leaving part of the substitution layer as the identity mapping. Despite concerns raised regarding this strategy [WWGY13], we will show that security is viable. To reach security in spite of a low MC, pseudorandomly generated binary matrices are used in the linear layer to introduce a very high degree of diffusion. A method to accountably instantiate LowMC is given in Section 3.3.

Encryption with LowMC starts with a key whitening, followed by several rounds of encryption where the exact number of rounds depends on the chosen parameter set. A single round is composed as follows:

$$\text{LOWMCRound}(i) = \\ \text{KEYADDITION}(i) \circ \text{CONSTANTADDITION}(i) \circ \text{LINEARLAYER}(i) \circ \text{SBOXLAYER}$$

In the following we give a detailed description of the individual steps.

SBOXLAYER is an m -fold parallel application of the same 3-bit Sbox on the first $3m$ bits of the state. If $n > 3m$ then for the remaining $n - 3m$ bits, the SboxLayer is the identity. The selection criteria for the Sbox were as follows:

¹ The number of Sboxes is limited though by the block size as the Sboxes need to fit into a block.

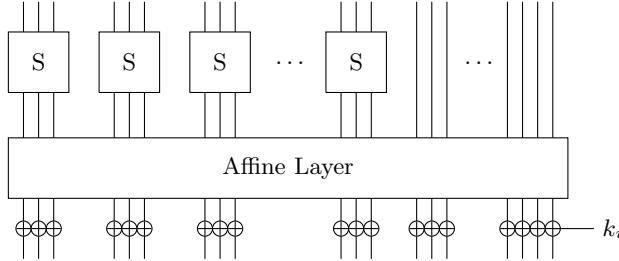


Fig. 1. Depiction of one round of encryption with LowMC

- Maximum differential probability: 2^{-2}
- Maximum linear probability: 2^{-2}
- Simple circuit description involving $MC = 3$ AND gates, with $ANDdepth=1$
- Each of the 8 non-zero component functions has algebraic degree 2

The Sbox is specified in 2, and coincides with the Sbox used for PRINTcipher [KLPR10]. Other representations of the Sbox can be found in the full version of this paper.

$\text{LINEARLAYER}(i)$ is the multiplication in $\text{GF}(2)$ of the state with the binary $n \times n$ matrix $\text{Lmatrix}[i]$. The matrices are chosen independently and uniformly at random from all invertible binary $n \times n$ matrices.

$\text{CONSTANTADDITION}(i)$ is the addition in $\text{GF}(2)$ of $\text{roundconstant}[i]$ to the state. The constants are chosen independently and uniformly at random from all binary vectors of length n .

$\text{KEYADDITION}(i)$ is the addition in $\text{GF}(2)$ of $\text{roundkey}[i]$ to the state. To generate $\text{roundkey}[i]$, the master key key is multiplied in $\text{GF}(2)$ with the binary $n \times k$ matrix $\text{Kmatrix}[i]$. The matrices are chosen independently and uniformly at random from all binary $n \times k$ matrices of rank $\min(n, k)$.

Decryption is done in the straightforward manner by an inversion of these steps.

$$S(a, b, c) = (a \oplus bc, a \oplus b \oplus ac, a \oplus b \oplus c \oplus ab)$$

Fig. 2. Specification of the 3-bit Sbox

3.1 Pseudocode

`plaintext` and `state` are n -bit quantities. `key` is a k -bit quantity, which can both be larger or smaller than n . `r` is the number of rounds.

```

ciphertext = encrypt (plaintext,key)
//initial whitening
state = plaintext + MultiplyWithGF2Matrix(KMatrix(0),key)

for (i = 1 to r)
    //m computations of 3-bit sbox,
    //remaining n-3m bits remain the same
    state = Sboxlayer (state)

    //affine layer
    state = MultiplyWithGF2Matrix(LMatrix(i),state)
    state = state + Constants(i)

    //generate round key and add to the state
    state = state + MultiplyWithGF2Matrix(KMatrix(i),state)
end
ciphertext = state

```

3.2 Parameters

Our security analysis against differential, linear, higher-order, meet-in-the-middle, algebraic, and slide attacks suggests that, except with negligible probability, any uniformly randomly chosen set of matrices leads to a secure construction for the parameters given in Table 1. For a larger selection of parameters bundled with security bounds, see the full version of this paper.

Table 1. Parameter sets of LowMC instantiations. One first set has PRESENT-like security parameters, the second set has AES-like security parameters.

blocksize	sboxes	keysize	data	rounds	ANDdepth	ANDs per bit
n	m	k	d	r		
256	49	80	64	11	11	6.3
256	63	128	128	12	12	8.86

3.3 Instantiation of LowMC

To maximize the amount of diffusion done by the linear layer, we rely on randomly generated, invertible binary matrices. As there exist no binary matrices of size larger than 1×1 that are MDS, and as it is generally an NP-complete problem to determine the branching number of a binary matrix [BMvT78], there is no obviously better method to reach this goal. The problem in the instantiation of LowMC is to find an accountable way of constructing the random matrices and vectors that leaves no room for the designer to plant backdoors.

Our recommended instantiation is a compromise between randomness, accountability and ease of implementation. It uses the Grain LSFR as a self-shrinking generator (see [HJMM08] and [MS94]) as a source of random bits. The exact procedure can be found in the full version of this paper.

It must be mentioned though that it is principally possible to use any sufficiently random source to generate the matrices and constants. It is also not necessary that the source is cryptographically secure.

4 Comparison with Other Ciphers

In the following we survey a larger number of existing cipher designs and study their ANDdepth and MC per encrypted bit which we summarize in Table 2. We both choose representative candidates from various design strategies, as well as the designs that are most competitive in terms of our metrics. We do this in two distinct categories: AES-like security (with key sizes of 128-bits and more and data security and block size of 128-bits and more), and lightweight security (data security and block size of 96 bits or below). Note that data security refers to the \log_2 of the allowable data complexity up to which a cipher is expected to give the claimed security against shortcut attacks. For LowMC we explicitly de-couple the data security from the block size of the cipher as the proposed design strategy favour larger block sizes but we don't see a new for larger data security than 128. For size-optimized variants we instantiate ℓ -bit adders using a ripple-carry adder which has $\ell - 1$ ANDs and ANDdepth $\ell - 1$; for depth-optimized variants we instantiate them with a Ladner-Fischer adder that has $\ell + 1.25\ell \log_2 \ell$ ANDs and ANDdepth $1 + 2\log_2 \ell$, cf. [SZ13].

We first survey AES versions and then ciphers with related security properties. The Sbox construction of [BP12] has 34 AND gates and ANDdepth 4 (the size optimized Sbox construction of [BMP13] has only 32 AND gates, but higher ANDdepth 6). See also Canright [Can05]. To encrypt a 128-bit block, AES-128 has 10 rounds and uses 160 calls to the Sbox (40 for key schedule), hence 5 440 AND gates, or 42.5 AND gates per encrypted bit. To encrypt a 128-bit block, AES-192 has 12 rounds and uses 192 calls to the Sbox (32 for key schedule), hence 6 528 AND gates, or 51 AND gates per encrypted bit. To encrypt a 128-bit block, AES-256 has 14 rounds and uses 224 calls to the Sbox (56 for key schedule), hence 7 616 AND gates, or 59.5 AND gates per encrypted bit.

AES is actually comparatively efficient. Other ciphers with a different design strategy can have very different properties. Threefish [FLS+10] is a cipher with large block size. Threefish with its 512-bit block size has 72 rounds with 4 additions modulo 2^{64} each resulting in 35.438 AND gates per encrypted bit and ANDdepth=4 536 (63 per round). Threefish with its 1 024-bit block size has 80 rounds with 8 additions each resulting in 39.375 AND gates per bit and ANDdepth=5 040 (63 per round). The recently proposed NSA cipher Simon [BSS+13] is also a good candidate to be of low multiplicative complexity. If b is the block size, it does $b/2$ AND gates per round, and ANDdepth is equal to the number of rounds. For a key size of 128 bit (comparable to AES) and block size 128 bit, it needs 68 rounds. This means, 4 352 AND gates, or 34 AND gates per bit.

In the lightweight category, we consider Present, but also Simon. The Present Sbox can be implemented with as little as 4 AND gates which is optimal [CHM11] and has ANDdepth 3. With $16 \cdot 31 = 496$ Sbox applications per 64 bit block we arrive at 31 AND gates per bit. A depth-optimized version of the Present Sbox with ANDdepth 2 and 8 ANDs is given in the full version of this paper. The 128bit secure version of Present differs only in the key schedule. Simon-64/96 has a 96 bit key, block size 64 bit and 42 rounds and Simon-32/64 has a 64 bit key, block size 32 bit and 32 rounds; see above for MC and ANDdepth. As another data point, the DES circuit of [TS] has 18175 AND gates and ANDdepth 261. KATAN [CDK09] has 254 rounds. In KATAN32, the ANDdepth increases by two every 8 rounds resulting in an ANDdepth of 64; with 3 AND gates per round and a block size of 32 bit this results in 23.81 ANDs per bit, but similar to Simon-32/64 applications are limited due to the small block size. In KATAN48 and KATAN64 the ANDdepth increases by 2 every 7 rounds resulting in an ANDdepth of 74. KATAN48 has 6 ANDs per round and a block size of 48 bit resulting in 31.75 ANDs per bit. KATAN64 has 9 ANDs per round and a block size of 64 bit resulting in 35.72 ANDs per bit. Prince [BCG+12] has 12 rounds and each round can be implemented with 10 AND gates and ANDdepth 2, cf. [DSES14]. NOEKEON [DPVAR00] is a competitive block cipher with 16 rounds and each round applies 32 S-boxes consisting of 4 AND gates with ANDdepth 2 each.

LowMC is easily parameterizable to all these settings, see also Table 1 in Section 3. It has at most (if $3m = n$) one AND gate per bit per round which results, together with a moderate number of rounds to make it secure, in the lowest ANDdepth and lowest MC per encrypted bit, cf. Table 2.

5 Resistance Against Cryptanalytic Attacks

The number of rounds r equals ANDdepth, and is hence a crucial factor to minimize. For this we evaluate the security of the construction against an array of known attack vectors. Below we especially discuss differential, linear and high-order attacks, as their analysis is a relevant technical contribution in itself. For a short discussion of other attack vectors, we refer to the full version of this paper.

We aim to prove the LowMC designs secure against classes of known attacks. However, due to the choice of random linear layers it is not immediately clear how to bound the probability of differential or linear characteristics. This is something we will investigate and resolve in Section 5.1. Due to the extremely simple description of the Sbox, higher order [Knu94] and cube attacks [DS09] that exploit a relatively slow growth in the algebraic degree appear to be the most promising attack vector, and are studied in Section 5.4. The quality of these bounds is tested on small versions of LowMC. This all will allow us to formulate in Section 5.6 a relatively simple expression for deriving a lower bound for the number of rounds given other parameters like the desired security level in terms of time and data, and block size.

Table 2. Comparison of ciphers (excluding key schedule). We list the depth-optimized variants; size-optimized variants are given in () if available. Best in class are marked in bold.

Cipher	Key size	Block size	Data sec.	ANDdepth	ANDs/bit	Sbox representation
AES-like security						
AES-128	128	128	128	40 (60)	43 (40)	[BP12] ([BMP13])
AES-192	192	128	128	48 (72)	51 (48)	[BP12] ([BMP13])
AES-256	256	128	128	56 (84)	60 (56)	[BP12] ([BMP13])
Simon	128	128	128	68	34	[BSS+13]
Simon	192	128	128	69	35	[BSS+13]
Simon	256	128	128	72	36	[BSS+13]
Noekeon	128	128	128	32	16	[DPVAR00]
Robin	128	128	128	96	24	[GLSV14]
Fantomas	128	128	128	48	16.5	[GLSV14]
Threefish	512	512	512	936 (4 536)	306 (36)	[FLS+10]
Threefish	512	1 024	1 024	1 040 (5 040)	340 (40)	[FLS+10]
LowMC	128	256	128	12	8.85	full version
Lightweight security						
PrintCipher-96	160	96	96	96	96	full version
PrintCipher-48	80	48	48	48	48	full version
Present	80 or 128	64	64	62 (93)	62 (31)	full version ([CHM11])
Simon	96	64	64	42	21	[BSS+13]
Simon	64	32	32	32	16	[BSS+13]
Prince	128	64	64	24	30	[DSES14]
KATAN64	80	64	64	74	36	[CDK09]
KATAN48	80	48	48	74	32	[CDK09]
KATAN32	80	32	32	64	24	[CDK09]
DES	56	64	56	261	284	[TS]
LowMC	80	256	64	11	6.31	full version

5.1 Differential Characteristics

In differential attacks, the principal goal is to find a pair (α, β) of an input difference α and an output difference β for the cipher such that pairs of input texts with difference α have an unusual high probability to produce output texts with difference β . Such a pair of differences is called a *differential*. A good differential can be used to mount distinguishing attacks as well as key recovery attacks on the cipher. For this it suffices if the differential does not cover the whole cipher but all except one or a few rounds.

As it is infeasible to calculate the probability of differentials for most ciphers, the cryptanalyst often has to be content with finding good *differential characteristics* i.e., paths of differences through the cipher for which the probability can directly be calculated. Note that a differential is made up of all differential characteristics that have the same input and output difference as the differential. The probability of a good differential characteristic is thus a lower bound for the related differential.

Allowing parts of the state to go unchanged through the Sbox layer clearly increases the chance of good differential characteristics. It is for example always possible to find a one round characteristic of probability 1. In fact, it is even possible to find $\lceil \frac{l}{3m} \rceil$ -round characteristics of probability 1 where l is the width of the identity part and m the number of 3-bit Sboxes. Nonetheless, as we will prove in the following, this poses no threat. This is because of the randomness of

the linear layer which maps a fixed subspace to a random subspace of the same dimension: Most “good” difference i.e., differences that activate none or only few Sboxes, are mapped to “bad” differences that activate most of the Sboxes per layer. This causes the number of characteristics that only use “good” differences to decay exponentially with the number of rounds. In the case of a $\lceil \frac{l}{3m} \rceil$ -round characteristic of probability 1, this means that the output difference is fixed to very few options, which makes it then already in the next round extremely unlikely that any one of the options is mapped onto a “good” difference.

We will now prove that good differential characteristics exist only with negligible probability in LowMC. The basic idea behind the proof is the following. We calculate for each possible good differential characteristic the probability that it is realized in an instantiation of LowMC under the assumption that the binary matrices of the linear layer were chosen independently and uniformly at random. We then show that the sum of these probabilities, which is an upper bound for the probability that any good characteristic exists, is negligible.

Recall that m is the number of Sboxes in one Sbox layer in LowMC and that l is the bit-length of the identity part of the Sbox layer. We thus have $n = 3m + l$. Let $V(i)$ be the number of bit vectors of length n that correspond to a difference that activates i Sboxes. As we can choose i out of the m Sboxes, as for each active 3-bit Sbox there are 7 possible non-zero input differences and as the bits of the identity part can be chosen freely, we have

$$V(i) = \binom{m}{i} \cdot 7^i \cdot 2^l . \quad (1)$$

Let α_0 be an input difference and let α_1 be an output difference for one round of LowMC. Let a_0 be the number of Sboxes activated by α_0 . As an active Sbox maps its non-zero input difference to four possible output differences each with probability $\frac{1}{4}$, and as a uniformly randomly chosen invertible binary $n \times n$ matrix maps a given non-zero n -bit vector with probability $\frac{1}{2^n - 1}$ to another given non-zero output vector, the probability that the one-round characteristic (α_0, α_1) has a probability larger than 0 is

$$\frac{4^{a_0}}{2^n - 1} . \quad (2)$$

Let $(\alpha_0, \alpha_1, \dots, \alpha_r)$ now be a given characteristic over r rounds where the differences α_i are at the end of round i and α_0 is the starting difference. Let $(a_0, a_1, \dots, a_{r-1})$ be the numbers of Sboxes activated by each $\alpha_0, \alpha_1, \dots$, and α_{r-1} . We can now calculate the probability that this characteristic has a probability larger than 0 in a random instantiation of LowMC as

$$\frac{4^{a_0}}{2^n - 1} \cdot \frac{4^{a_1}}{2^n - 1} \cdots \frac{4^{a_{r-1}}}{2^n - 1} = \frac{4^{a_0 + a_1 + \cdots + a_{r-1}}}{(2^n - 1)^r} . \quad (3)$$

Summing now over all possible characteristics over r rounds that activate at most d Sboxes, we can calculate an upper bound for the probability that there exists an r -round characteristic with d or fewer active Sboxes as

$$\sum_{\substack{0 \leq a_0, a_1, \dots, a_{r-1} \leq m \\ a_0 + a_1 + \dots + a_{r-1} \leq d}} V(a_0) \cdot V(a_1) \cdots V(a_{r-1}) \cdot (2^n - 1) \cdot \frac{4^{a_0 + a_1 + \dots + a_{r-1}}}{(2^n - 1)^r} \quad (4)$$

where the factor $(2^n - 1)$ is the number of choices for the last difference α_r that can take any non-zero value.

With the knowledge that each active Sbox reduces the probability of a characteristic by a factor of 2^{-2} , we can now calculate for each parameter set of LowMC the number of rounds after which no good differentials are present except for a negligible probability. We consider as good differential characteristics those with a probability higher than 2^{-d} , where d is the allowed data complexity in the respective parameter set. We call a negligible probability a probability lower than 2^{-100} . Note that this probability only comes into play once when fixing an instantiation of LowMC. The calculated bound for our choice of parameters can be found in Table 4.

Table 3. Example of how the probability bound p_{stat} , for the existence of differential or linear characteristic of probability at least 2^{-d} , evolves. The parameters are here $m = 42$, $d = 128$.

Rounds	1 - 6	7	8	9	10	11	12	13	14	15
$n = 256$	1.0	2^{-100}	2^{-212}	2^{-326}	2^{-442}	2^{-558}	2^{-676}	2^{-794}	2^{-913}	-
$n = 1024$	1.0	1.0	1.0	1.0	1.0	1.0	2^{-26}	2^{-145}	2^{-264}	

5.2 Linear Characteristics

In linear cryptanalysis [Mat93], the goal of the cryptanalyst is to find affine approximations of the cipher that hold sufficiently well. As with differential cryptanalysis, these can be used to mount distinguishing and key recovery attacks. The approximation is done by finding so-called *linear characteristics*, a concatenation of linear approximations for the consecutive rounds of the cipher. Similar to differential characteristics, linear characteristics activate Sboxes that are involved in the approximations.

The proof for the absence of good differential characteristics is directly transferable to linear characteristics because of two facts. Firstly, the maximal linear probability of the Sbox is 2^{-2} , just the same as the maximal differential probability. Secondly, the transpose of a uniformly randomly chosen invertible binary matrix is still a uniformly randomly chosen invertible binary matrix. Thus we can use equation 4 to calculate the bounds for good linear characteristics as well.

5.3 Boomerang Attacks

In boomerang attacks [Wag99], good partial differential characteristics that cover only part of the cipher can be combined to attack ciphers that might be immune

to standard differential cryptanalysis. In these attacks, two differential characteristics are combined, one that covers the first half of the cipher and another that covers the second half. If both have about the same probability, the complexity corresponds roughly to the inverse of the fourth power of this probability [Wag99]. Thus to calculate the number of rounds sufficient to make sure that no boomerang exists, we calculate the number of rounds after which differential characteristics of probability $2^{-d/4}$ exist only with negligible probability and then double this number.

5.4 Higher Order Attacks

Due to its small size, the degree of the Sbox in its algebraic representation is only two. Since in one round the Sboxes are applied in parallel and since the affine layer does not change the algebraic degree, the algebraic degree of one round is two as well. As a low degree could be used as a lever for a high-order attack, let us take a look at how the algebraic degree of LowMC develops over several rounds.

Clearly the algebraic degree of the cipher after r rounds is bounded from above by 2^r . It is furthermore generally bounded from above by $n - 1$ since the cipher is a permutation. A second upper bound, that is better suited and certainly more realistic for the later rounds, was found by Boura et al. [BCC11]. In our case it is stated as following: If the cipher has degree d_r after r rounds, the degree after round $r + 1$ is at most $\frac{n}{2} + \frac{d_r}{2}$. Differing from Boura et al. [BCC11], in LowMC the Sbox layer only partially consists of Sboxes and partially of the identity mapping. This must be accounted for and requires a third bound: If the cipher has degree d_r after r rounds, the degree after round $r + 1$ is at most $m + d_r$. A proof of this can be found in the full version of this paper. This can be summarized as follows:

Lemma 1. *If the algebraic degree of LowMC with m Sboxes and length l of the identity part in the Sbox layer is d_r after r rounds, the degree in round $r + 1$ is at most*

$$\min \left(2d_r, m + d_r, \frac{n}{2} + \frac{d_r}{2} \right) \quad (5)$$

where $n = 3m + l$ is the block width of LowMC.

Combining these three bounds, we can easily calculate lower bounds for the number of rounds r needed for different parameter sets l and m of LowMC to reach a degree that is at least as large as the allowed data complexity d minus 1. The results of this for LowMC's parameters are displayed in Table 4.

5.5 Experimental Cryptanalysis

We proved that no good differential or linear characteristic can cover sufficiently many rounds to be usable as an attack vector in LowMC. This does not exclude

Table 4. For the different sets of LowMC parameters, bounds are given for the number of rounds for which no good differential or linear characteristics exist (r_{stat}), to avoid good boomerangs (r_{bmrg}), and the number of rounds needed to have a sufficiently high algebraic degree (r_{deg}). The bounds were calculated using equations 4 and 5.

Sboxes	blocksize	data complexity	r_{stat}	r_{bmrg}	r_{deg}
49	256	64	5	6	6
63	256	128	5	6	7

though the possibility of good differentials or linear hulls for which a large number of characteristics combine. Given the highly diffusive, random linear layers, this seems very unlikely.

Likewise we were able to find lower bounds on the number of rounds needed for the algebraic degree of LowMC to be sufficiently high. Even though this is state-of-the art also for traditional designs to date, this gives us no guarantee that it will indeed be high. Unfortunately it is not possible to directly calculate the algebraic degree for any large block size.

To nevertheless strengthen our confidence in the design, we numerically examined the properties of small-scale versions of LowMC. In table 5, we find the results for a 24-bit wide version with 4 Sboxes. For testing its resistance against differential cryptanalysis, we calculated the full codebook under 100 randomly chosen keys and used the distribution of differences to estimate the probabilities of the differentials. To reduce the computational complexity, we restricted the search space to differentials with one active bit in the input difference.

It can clearly be seen that the probability of differentials quickly saturates to values too low to allow an attack. Clearly, the bound calculated with equation 4 (p_{stat} in the table) overestimates the probability of good characteristics. Even though we were not able to search the whole space of differentials there is little reason to assume that there are other differentials that fare considerably better. It is important to note that the number of impossible differentials goes to 0 after only few rounds. Thus impossible differentials cannot be used to attack any relevant number of rounds. At the same time this assures the absence of any truncated differentials of probability 1.

The minimal algebraic degree² is tight for this version when compared with the theoretic upper bound as determined with equation 5. More experimental cryptanalysis can be found in the full version of this paper.

5.6 Fixing the Number of Rounds

We base our recommendation for the number of rounds on the following:

$$r \geq \max(r_{\text{stat}}, r_{\text{bmrg}}, r_{\text{deg}}) + r_{\text{outer}}$$

² That is the minimum of the algebraic degrees of the 24 output bit when written as Boolean functions.

Table 5. Experimental results of full codebook encryption over 100 random keys for a set of small parameters are given. p_{best} and p_{worst} are the best and the worst approximate differential probability of any differential with one active bit in the input difference. n_{imposs} is the number of impossible differentials with one active bit in the input difference. \deg_{exp} is the minimal algebraic degree in any of the output bits. \deg_{theor} is the upper bound for the algebraic degree as determined from equation 5. p_{stat} is the probability that a differential or linear characteristic of probability at least 2^{-12} exists (see eq. 4).

(a) $n = 24, m = 4, k = 12, d = 12$

Rounds	p_{best}	p_{worst}	n_{imposs}	\deg_{exp}	\deg_{theor}	p_{stat}
2	$2^{-8.64}$	0	$2^{28.58}$	4	4	-
3	$2^{-12.64}$	0	$2^{28.00}$	8	8	-
4	$2^{-14.64}$	0	$2^{4.25}$	12	12	-
5	$2^{-18.60}$	$2^{-26.06}$	0	16	16	-
6	$2^{-20.49}$	$2^{-25.84}$	0	20	20	-
7	$2^{-23.03}$	$2^{-25.74}$	0	22	22	-
8	$2^{-23.06}$	$2^{-25.74}$	0	23	23	-
10	-	-	-	-	-	$2^{-5.91}$
11	-	-	-	-	-	$2^{-16.00}$
12	-	-	-	-	-	$2^{-26.28}$
19	-	-	-	-	-	$2^{-101.5}$

where r_{stat} is a bound for statistical attack vectors such as differentials and linear characteristics as discussed in Section 5.1, r_{bmrng} is the bound for boomerang attacks as discussed in Section 5.3, and where r_{deg} indicates the number of rounds needed for the cipher to have sufficient degree as discussed in Section 5.4. Values of these for the parameters of LowMC can be found in Table 4. For the number of rounds which can be peeled off at the beginning and end of the cipher by key guessing and other strategies, we use the ad-hoc formula $r_{\text{outer}} = r_{\text{stat}}$.

6 Comparison of Implementations

In the following we report on experiments when evaluating LowMC with MPC protocols in Section 6.1 and with FHE in Section 6.2. The performance of both implementations is independent of the specific choice of the random matrices and vectors used in LowMC (cf. Section 3.3) as we do not use any optimizations that are based on their specific structure.

In both the FHE and MPC settings, for more efficient matrix multiplication, we use a method that is generically better than a naive approach: the “method of the four Russians” [ABH10]. This method reduces the complexity of the matrix-vector product from $O(n^2)$ to $O(n^2/\log(n))$, i.e. it’s an asymptotically faster algorithm and is also fast in practice for the dimensions we face in LowMC.

Asymptotically faster methods like the Strassen-Winograd method make no sense however, for the dimensions we are considering.

It turns out that considering design-optimizations of the linear layer by introducing structure and thereby lowering the density of the matrices and in turn reducing the number of XOR computations will not improve performance of all these implementations. On the contrary, as the application of the security analysis suggests, the number of rounds would need to be increased in such a case.

6.1 MPC Setting

As an example for both classes of MPC protocols described in Section 2.1 we use the GMW protocol [GMW87] in the semi-honest setting. As described in [CHK+12], this protocol can be partitioned into 1) a *setup phase* with a constant number of rounds and communication linear in the MC of the circuit (2κ bits per AND gate for κ -bit security), and 2) an *online phase* whose round complexity is linear in the ANDdepth of the circuit. Hence, we expect that the setup time grows linearly in the MC while the online time grows mostly with increasing ANDdepth when network latency is high.

Benchmark Settings. For our MPC experiments we compare LowMC against other ciphers with a comparable level of security. We compare LowMC with the two standardized ciphers Present and AES and also with the NSA cipher Simon which previously had the lowest number of ANDs per encrypted bit (cf. Table 2). More specifically, for lightweight security with at least $\kappa = 80$ bit security we compare LowMC with 80 bit keys against Present with 80 bit key (using the Sbox of [CHM11]) and Simon with 96 bit keys (the Simon specification does not include a variant with 80 bit keys); for long-term security with $\kappa = 128$ bit security we compare LowMC with 128 bit keys against AES-128 (using the Sbox of [BP12]) and Simon with 128 bit key; we set the security parameters for the underlying MPC protocol to $\kappa = 80$ bit for lightweight security and to $\kappa = 128$ bit for long-term security. We exclude the key schedule and directly input the pre-computed round keys. We use the GMW implementation that is available in the ABY-framework [DSZ15] which uses the efficient oblivious transfer extensions of [ALSZ13]³. We run our MPC experiments on two desktop PCs, each equipped with an Intel Haswell i7-4770K CPU with 3.5 GHz and 16GB of RAM, that are connected by Gigabit LAN. To see the impact of the reduced ANDdepth in the online phase, we measured the times in a LAN scenario (0.2 ms latency) and also a trans-atlantic WAN scenario (50 ms latency) which we simulated using the Linux command `tc`.

In our first experiment depicted in Table 6 we encrypt a single block, whereas in our second experiment depicted in Table 7 we encrypt multiple blocks in parallel to encrypt 12.8 Mbit of data.

³ Our MPC implementations of the benchmarked block-ciphers are available online as part of the ABY-framework <https://github.com/encryptogroup/ABY>.

Table 6. GMW benchmarking results for single block. Best in class marked in bold.

Lightweight Security						
Cipher	Present		Simon		LowMC	
Communication [kB]	39		26		51	
Runtime	LAN	WAN	LAN	WAN	LAN	WAN
Setup [s]	0.003	0.21	0.002	0.21	0.002	0.14
Online [s]	0.05	13.86	0.05	5.34	0.06	1.46
Total [s]	0.05	14.07	0.05	5.45	0.06	1.61

Long-Term Security						
Cipher	AES		Simon		LowMC	
Communication [kB]	170		136		72	
Runtime	LAN	WAN	LAN	WAN	LAN	WAN
Setup [s]	0.01	0.27	0.009	0.23	0.002	0.15
Online [s]	0.04	4.08	0.05	6.95	0.07	1.87
Total [s]	0.05	4.35	0.06	7.18	0.07	2.02

Single-Block Results. From our single-block experiments in Table 6 we see that the communication of LowMC is higher by factor 2 compared to the lightweight security ciphers but lower by factor 2 compared to the long-term security ciphers. In terms of total runtime, for lightweight security LowMC performs similar to Present and Simon in the LAN setting and outperforms both by factor 3 to 9 in the WAN setting. For long-term security AES is slightly faster than LowMC in the LAN setting, but slower than LowMC in the WAN setting by factor 2. These results can be explained by the high number of XOR gates of LowMC compared to AES, which impact the run-time higher than the communication for the AND gates. In the WAN setting, the higher ANDdepth of AES outweighs the local overhead of the XOR gates for LowMC, yielding a faster run-time for LowMC.

Multi-Block Results. From our multi-block experiments in Table 7 we see that LowMC needs less communication than all other ciphers: at least factor 2 for lightweight security and factor 4 for long-term security. Also the total runtime of LowMC is the lowest among all ciphers, ranging from factor 6 when compared to Simon for lightweight security to factor 9 when compared to AES for long-term security.

Summary of the Results. To summarize our MPC experiments, the benefits of LowMC w.r.t. the *online time* depend on the network latency: over the low-latency LAN network existing ciphers achieve comparable or even slightly faster online runtimes than LowMC, whereas in the higher latency WAN network LowMC achieves the fastest online runtime. W.r.t. the *total runtime*, LowMC’s benefit in the single-block application again depends on the latency (comparable or slightly less efficient over LAN, but more efficient over WAN), whereas in the multi-block application LowMC significantly improves over existing ciphers by factor 6 to 9. For secure computation protocols with security against malicious adversaries, the benefit of using LowMC would be even more significant, since

Table 7. GMW benchmarking results for multiple blocks to encrypt 12.8 Mbit of data. Best in class marked in bold.

<i>Lightweight Security</i>						
Cipher	Present		Simon		LowMC	
Comm. [GB]	7.4		5.0		2.5	
Runtime	LAN	WAN	LAN	WAN	LAN	WAN
Setup [s]	214.17	453.89	268.93	568.35	43.33	138.63
Online [s]	2.71	34.35	3.29	37.06	2.02	17.12
Total [s]	216.88	488.24	272.22	605.41	45.36	155.75
<i>Long-Term Security</i>						
Cipher	AES		Simon		LowMC	
Comm. [GB]	16		13		3.5	
Runtime	LAN	WAN	LAN	WAN	LAN	WAN
Setup [s]	553.41	914.27	444.30	727.48	62.01	193.90
Online [s]	2.50	33.52	2.97	34.42	2.36	21.11
Total [s]	555.91	947.79	447.27	761.90	64.37	215.01

there the costs per AND gate are at least an order of magnitude higher than in the semi-honest GMW protocol, cf. [NNOB12, LOS14].

6.2 FHE Setting

We implemented LowMC using the homomorphic encryption library HELib [HS13, HS14], which implements the BGV homomorphic encryption scheme [BGV11] and which was also used to evaluate AES-128 [GHS12a, GHS12b]. Our implementation represents each plaintext, ciphertext and key bits as individual HE ciphertexts on which XOR and AND operations are performed. Due to the nature of the BGV system this means that we can evaluate many such instances in parallel, typically a few hundred. We found this representation to be more efficient than our other “compact” implementation which packs these bits into the slots of HE ciphertexts.

In the homomorphic encryption setting the number of AND gates is not the main determinant of complexity. Instead, the ANDdepth of the circuit largely determines the cost of XOR and AND, where AND is more expensive than XOR. However, due to the high number of XORs in LowMC, the cost of the linear layer is not negligible. In our implementation we use the “method of the four Russians” [ABH10] to reduce the number of HE ciphertext additions from $\mathcal{O}(n^2)$ to $\mathcal{O}(n^2/\log(n))$.

In our experiments we chose the depth for the homomorphic encryption scheme such that the “base level” of fresh ciphertexts is at least the number of rounds, i.e. we consume one level per round. Our implementation also does not precompute round keys in advance, but deriving round keys is considered part of the evaluation (cost).

We consider LowMC instances for Present-80 and AES-128 like security. We always choose a homomorphic encryption security level of 80 for compatibility with [GHS12b]. Our results are given in Table 8. Our implementation is available at <https://bitbucket.org/malb/lowmc-helib>.

Table 8. LowMC (commit `f6a086e`) in HElib [HS13] (commit `e9d3785e`) on Intel i7-4850HQ CPU @ 2.30GHz; d is the allowed data complexity, m is the number of Sboxes, n is the blocksize, r is the number of rounds, # blocks is the number of blocks computed in parallel, t_{setup} is the total setup time, t_{eval} is the total running time of the encryption in seconds, t_{sbox} the total time spent in the S-Box layer in seconds, t_{key} the total time spent in the key schedule in seconds, $t_{\text{block}} = t_{\text{eval}}/\#\text{blocks}$ and $t_{\text{bit}} = t_{\text{block}}/n$. The rows marked as “main” contain the main parameter proposals. The rows marked as “perf”, “cons” or “smll” contain alternative parameter sets being conservative, performance oriented or relatively small respectively.

d	m	r	n	# blocks	t_{setup}	t_{eval}	t_{sbox}	t_{key}	t_{block}	t_{bit}	Memory	Comment
128	63	12	256	600	11.6	506.1	353.2	1.6	0.8434	0.0033	1.58GB	main
128	86	11	512	600	11.7	847.6	451.5	3.2	1.4127	0.0028	2.62GB	perf
128	86	12	512	600	11.7	893.9	480.1	3.2	1.4898	0.0029	2.62GB	cons
64	49	11	256	600	11.0	383.0	206.3	0.9	0.6383	0.0025	1.52GB	main
64	49	10	256	600	11.5	305.6	255.6	1.1	0.5093	0.0020	1.37GB	perf
64	34	11	128	600	13.0	260.7	204.0	0.7	0.4345	0.0034	1.08GB	smll

For comparison with previous results in the literature we reproduce those results in Table 9 which demonstrates the benefit of a dedicated block cipher for homomorphic evaluation.

Table 9. Comparison of various block cipher evaluations in the literature and this work; Notation as in Table 8. Memory requirements are not listed as they are usually not provided in the literature. The first row is based on experimental data obtained on the same machine and the same instance of HElib as in Table 8.

d	ANDdepth	# blocks	t_{eval}	t_{block}	t_{bit}	Cipher	Reference	Key Schedule
128	40	120	3m	1.5s	0.0119s	AES-128	[GHS12b]	excluded
128	40	2048	31h	55s	0.2580s	AES-128	[DHS14]	excluded
128	40	1	22m	22m	10.313s	AES-128	[MS13]	excluded
128	40	12	2h47m	14m	6.562s	AES-128	[MS13]	excluded
128	12	600	8m	0.8s	0.0033s	LowMC	this work	included
64	24	1024	57m	3.3s	0.0520s	PRINCE	[DSES14]	excluded
64	11	600	6.4m	0.64s	0.0025s	LowMC	this work	included

7 Conclusions, Lessons Learned, and Open Problems

We proposed block ciphers with an extremely small number of AND gates and an extremely shallow AND depth, demonstrated the soundness of our design through experimental evidence and provided a security analysis of these constructions. Of course, as with any other block cipher, more security analysis is needed to firmly establish the security provided by this new design. Furthermore, with the proposal of the LowMC family, we bring together the areas of symmetric cryptographic design and analysis research with new developments

around MPC and FHE. Finally, in contrast to current folklore belief, in some implementation scenarios, we identified practical cases where “free XORs” can no longer be considered free and where local computations in an MPC protocol represent a considerable bottleneck.

To finish, we highlight a number of open problems related to the LowMC family of ciphers. Is it possible to reduce the number of rounds in LowMC further, which in turn would further reduce MC and ANDdepth? Analyzing such an extreme corner of the design space for a symmetric cipher is an interesting endeavor in itself. Can we add more structure into the linear layers in order to reduce the necessary computational effort in those cases where the number of AND gates is no longer the bottleneck? Do such approaches beat applying asymptotically faster linear algebra techniques for applying linear layers as done in Section 6? As we argue in the paper, simply lowering the density of the matrices by several factors of two will not be enough.

Currently, the MC and ANDdepth of various cipher constructions is poorly understood. For example, it would be interesting to find efficient algorithms along the lines of [BMP13] for the various ciphers including the recent lightweight cipher proposals in the literature. While our choice for GF(2) is well motivated, there are scenarios where larger fields might be beneficial. What designs minimize MC and ANDdepth under such constraints?

Acknowledgments. We thank Dmitry Khovratovich for pointing to us out that an earlier version of our parameter sets for LowMC instantiations are too optimistic. See the full version of this paper for details. We thank Orr Dunkelman for helpful clarifications on the cipher KATAN. We thank Gaëtan Leurent and François-Xavier Standaert for helpful clarifications regarding the ciphers Fantomas and Robin.

The work of Albrecht was supported by EPSRC grant EP/L018543/1 “Multilinear Maps in Cryptography”. The work of co-authors from TU Darmstadt was supported by the European Union’s 7th Framework Program (FP7/2007-2013) under grant agreement n. 609611 (PRACTICE), by the DFG as part of project E3 within the CRC 1119 CROSSING, by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE, and by the Hessian LOEWE excellence initiative within CASED.

References

- [ABH10] Albrecht, M.R., Bard, G.V., Hart, W.: Algorithm 898: Efficient multiplication of dense matrices over GF(2). *ACM Transactions on Mathematical Software* **37**(1) (2010)
- [AIK06] Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC⁰. *SIAM Journal on Computing* **36**(4), 845–888 (2006)
- [AL07] Aumann, Y., Lindell, Y.: Security against covert adversaries: efficient protocols for realistic adversaries. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 137–156. Springer, Heidelberg (2007)
- [ALSZ13] Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: Computer and Communications Security (CCS), pp. 535–548. ACM (2013). Code: <http://github.com/MichaelZohner/OTExtension>

- [BC86] Brassard, G., Crépeau, C.: Zero-knowledge simulation of boolean circuits. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 223–233. Springer, Heidelberg (1987)
- [BCC11] Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of KECCAK and *Luffa*. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011)
- [BCG+12] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE – a low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
- [BDP00] Boyar, J., Damgård, I., Peralta, R.: Short non-interactive cryptographic proofs. *Journal of Cryptology* **13**(4), 449–472 (2000)
- [BGV11] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)* **18**, 111 (2011)
- [BMP13] Boyar, J., Matthews, P., Peralta, R.: Logic minimization techniques with applications to cryptology. *Journal of Cryptology* **26**(2), 280–312 (2013)
- [BMvT78] Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.A.: On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory* **24**(3), 384–386 (1978)
- [BODD+14] Bar-On, A., Dinur, I., Dunkelman, O., Lallemand, V., Keller, N., Tsabani, B.: Cryptanalysis of SP Networks with Partial Non-Linear Layers. In: Cryptology ePrint Archive, Report 2014/228 (2014). <http://eprint.iacr.org/2014/228>
- [BP12] Boyar, J., Peralta, R.: A small depth-16 circuit for the AES S-box. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IFIP AICT, vol. 376, pp. 287–298. Springer, Heidelberg (2012)
- [BPP00] Boyar, J., Peralta, R., Pochuev, D.: On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. *Theoretical Computer Science* **235**(1), 43–57 (2000)
- [BSS+13] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. In: Cryptology ePrint Archive, Report 2013/404 (2013). <http://eprint.iacr.org/2013/404>
- [Can05] Canright, D.: A very compact S-box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005)
- [CDK09] De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
- [CHK+12] Choi, S.G., Hwang, K.-W., Katz, J., Malkin, T., Rubenstein, D.: Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 416–432. Springer, Heidelberg (2012). <http://www.ee.columbia.edu/~kwhwang/projects/gmw.html>
- [CHM11] Courtois, N.T., Hulme, D., Mourouzis, T.: Solving circuit optimisation problems in cryptography and cryptanalysis. In: Cryptology ePrint Archive, Report 2011/475 (2011). <http://eprint.iacr.org/2011/475>

- [DHS14] Doroz, Y., Hu, Y., Sunar, B.: Homomorphic AES evaluation using NTRU. In: Cryptology ePrint Archive, Report 2014/039 (2014). <http://eprint.iacr.org/2014/039>
- [DLT14] Damgård, I., Lauritsen, R., Toft, T.: An empirical study and some improvements of the minimac protocol for secure computation. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 398–415. Springer, Heidelberg (2014)
- [DPVAR00] Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie proposal: Noekeon. In: First Open NESSIE Workshop (2000)
- [DS09] Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
- [DSES14] Doröz, Y., Shahverdi, A., Eisenbarth, T., Sunar, B.: Toward practical homomorphic evaluation of block ciphers using Prince. In: Cryptology ePrint Archive, Report 2014/233 (2014), presented at Workshop on Applied Homomorphic Cryptography and Encrypted Computing (WAHC 2014). <http://eprint.iacr.org/2014/233>
- [DSZ15] Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: Network and Distributed System Security, NDSS 2015. The Internet Society (2015). Code: <https://github.com/encryptogroup/ABY>
- [DZ13] Damgård, I., Zakarias, S.: Constant-overhead secure computation of boolean circuits using preprocessing. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 621–641. Springer, Heidelberg (2013)
- [FJN+13] Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B., Nordholt, P.S., Orlandi, C.: MiniLEGO: efficient secure two-party computation from general assumptions. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 537–556. Springer, Heidelberg (2013)
- [FJN14] Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B.: Faster maliciously secure two-party computation using the GPU. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 358–379. Springer, Heidelberg (2014)
- [FLS+10] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to NIST (Round 3) (2010)
- [FN13] Frederiksen, T.K., Nielsen, J.B.: Fast and maliciously secure two-party computation using the GPU. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 339–356. Springer, Heidelberg (2013)
- [GGNPS13] Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.-X.: Block ciphers that are easier to mask: how far can we go? In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 383–399. Springer, Heidelberg (2013)
- [GHS12a] Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012)
- [GHS12b] Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Cryptology ePrint Archive, Report 2012/099 (2012). <http://eprint.iacr.org/2012/099>

- [GLSV14] Gross, V., Leurent, G., Standaert, F.-X., Varici, K.: LS-designs: Bitslice encryption for efficient masked software implementations. In: FSE 2014. LNCS, vol. 8540. Springer, Heidelberg (2015)
- [GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Symposium on Theory of Computing (STOC), pp. 218–229. ACM (1987)
- [GNPW13] Guo, J., Nikolic, I., Peyrin, T., Wang, L.: Cryptanalysis of Zorro. In: Cryptology ePrint Archive, Report 2013/713 (2013). <http://eprint.iacr.org/2013/713>
- [HEKM11] Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security. USENIX (2011)
- [HJMM08] Hell, M., Johansson, T., Maximov, A., Meier, W.: The grain family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 179–190. Springer, Heidelberg (2008)
- [HKE13] Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 18–35. Springer, Heidelberg (2013)
- [HKK+14] Huang, Y., Katz, J., Kolesnikov, V., Kumaresan, R., Malozemoff, A.J.: Amortizing garbled circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 458–475. Springer, Heidelberg (2014)
- [HS13] Halevi, S., Shoup, V.: Design and implementation of a homomorphic-encryption library (2013). <https://github.com/shaih/HElib/>
- [HS14] Halevi, S., Shoup, V.: Algorithms in HElib. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 554–571. Springer, Heidelberg (2014)
- [JKO13] Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Computer and Communications Security (CCS), pp. 955–966. ACM (2013)
- [KLPR10] Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINT-CIPHER: a block cipher for IC-printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010)
- [Knu94] Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
- [KS08] Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórssen, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
- [KSS12] Kreuter, B., Shelat, A., Shen, C.-H.: Billion-gate secure computation with malicious adversaries. In: USENIX Security. USENIX (2012)
- [Lin13] Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (2013)
- [LOS14] Larraia, E., Orsini, E., Smart, N.P.: Dishonest majority multi-party computation for binary circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 495–512. Springer, Heidelberg (2014)
- [LP07] Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)

- [LP11] Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer, Heidelberg (2011)
- [LPS08] Lindell, Y., Pinkas, B., Smart, N.P.: Implementing two-party computation efficiently with security against malicious adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 2–20. Springer, Heidelberg (2008)
- [LR14] Lindell, Y., Riva, B.: Cut-and-choose yao-based secure computation in the online/offline and batch settings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 476–494. Springer, Heidelberg (2014)
- [Mat93] Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
- [MNPS04] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay – a secure two-party computation system. In: USENIX Security, pp. 287–302. USENIX (2004)
- [MS94] Meier, W., Staffelbach, O.: The self-shrinking generator. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 205–214. Springer, Heidelberg (1995)
- [MS13] Mella, S., Susella, R.: On the homomorphic computation of symmetric cryptographic primitives. In: Stam, M. (ed.) IMACC 2013. LNCS, vol. 8308, pp. 28–44. Springer, Heidelberg (2013)
- [NNOB12] Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012)
- [PRC12] Piret, G., Roche, T., Carlet, C.: PICARO – a block cipher allowing efficient higher-order side-channel resistance. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 311–328. Springer, Heidelberg (2012)
- [PSSW09] Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
- [RASA14] Rasoolzadeh, S., Ahmadian, Z., Salmasizadeh, M., Aref, M.R.: Total Break of Zorro using Linear and Differential Attacks. In: Cryptology ePrint Archive, Report 2014/220 (2014). <http://eprint.iacr.org/2014/220>
- [SS11] Shelat, A., Shen, C.-H.: Two-Output Secure Computation with Malicious Adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer, Heidelberg (2011)
- [SS13] Shelat, A., Shen, C.-H.: Fast two-party secure computation with minimal assumptions. In: Computer and Communications Security (CCS), pp. 523–534. ACM (2013)
- [SZ13] Schneider, T., Zohner, M.: GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 275–292. Springer, Heidelberg (2013)
- [TS] Tillich, S., Smart, N.: Circuits of basic functions suitable for MPC and FHE. <http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/>
- [Wag99] Wagner, D.: The boomerang attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)

- [WWGY13] Wang, Y., Wu, W., Guo, Z., Yu, X.: Differential Cryptanalysis and Linear Distinguisher of Full-Round Zorro. In: Cryptology ePrint Archive, Report 2013/775 (2013). <http://eprint.iacr.org/2013/775>
- [Yao86] Yao, A.C.-C.: How to generate and exchange secrets. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 162–167. IEEE (1986)

Masking

Verified Proofs of Higher-Order Masking

Gilles Barthe¹(✉), Sonia Belaïd², François Dupressoir¹, Pierre-Alain Fouque³,
Benjamin Grégoire⁴, and Pierre-Yves Strub¹

¹ IMDEA Software Institute, Madrid, Spain

{gilles.barthe,francois.dupressoir,pierre-yves.strub}@imdea.org

² École normale supérieure and Thales Communications and Security,
Paris and Gennevilliers, France
sonia.belaïd@ens.fr

³ Université de Rennes 1 and Institut universitaire de France, Rennes, France

pierre-alain.fouque@ens.fr

⁴ INRIA, Sophia-Antipolis, France

benjamin.gregoire@inria.fr

Abstract. In this paper, we study the problem of automatically verifying higher-order masking countermeasures. This problem is important in practice, since weaknesses have been discovered in schemes that were thought secure, but is inherently exponential: for t -order masking, it involves proving that every subset of t intermediate variables is distributed independently of the secrets. Some tools have been proposed to help cryptographers check their proofs, but are often limited in scope.

We propose a new method, based on program verification techniques, to check the independence of sets of intermediate variables from some secrets. Our new language-based characterization of the problem also allows us to design and implement several algorithms that greatly reduce the number of sets of variables that need to be considered to prove this independence property on *all* valid adversary observations. The result of these algorithms is either a proof of security or a set of observations on which the independence property cannot be proved. We focus on AES implementations to check the validity of our algorithms. We also confirm the tool’s ability to give useful information when proofs fail, by rediscovering existing attacks and discovering new ones.

Keywords: Higher-order masking · Automatic tools · EasyCrypt

1 Introduction

Most widely used cryptographic algorithms are assumed to be secure in the *black-box* model, that is when the adversary is only given access to the inputs and outputs of the algorithm. However, this model does not fit the reality of embedded devices. In practice, an attacker can observe the physical leakage of a device in order to mount *side-channel attacks*. These attacks exploit the dependence between secret values used in the computation and the physical

leakage inherent to the physical implementation and execution (for example, timing, power consumption or electromagnetic radiations). Such attacks are very efficient in practice, with a recent attack recovering a full AES key using a single power trace [34]. Further, *differential attacks* can be mounted that exploit similar dependencies between *sensitive* values, that depend on both secret inputs and adversarially-controlled public inputs, to speed up the rate at which information on the secrets is gathered. Differential Power Analysis [22] (DPA), in particular, is a very effective class of attacks.

Masking. In order to thwart Differential Power Analysis, the community have proposed many countermeasures but *masking* remains most widely used. Masking makes use of a secret-sharing scheme to split each secret or sensitive variable into $(t + 1)$ shares such that the joint distribution of any subset of at most t shares is independent of the secret, but the knowledge of all $(t + 1)$ shares allows for the efficient recovery of the secret. The computation itself is then masked as well, replacing basic operations on, say, bytes with complex operations on $(t + 1)$ bytes. Intuitively, an implementation that is split over $(t + 1)$ shares should be able to resist the leakage of t of its intermediate variables (t is then usually called the *masking order*). Most of the implementations were then masked at order 1 to prevent an adversary from recovering secrets using a single observation. However, even higher-order attacks, where t is greater than 1 have been conducted in practice [26, 28] and need to be protected against. Many masked implementations have been proposed to protect AES or its non-linear component, the S-box (for example, [10, 27, 30, 31, 33]), among which some are also proved secure. Checking first-order masking schemes is a relatively routine task since it is sufficient to check that each intermediate variable carries a distribution that is independent from the secret. However, manually checking higher-order masked implementations is a more difficult and error-prone task. As a consequence, many published schemes were later shown to be insecure, such as those presented by [33] and [31], which were later broken in [12] and [13]. In this paper, we address this issue by developing automated methods to verify the security of algorithms masked at higher orders.

Adversary Models. The first step towards formally reasoning about the security of masked algorithms is to define a leakage model that formally captures the information that is leaked to the adversary. For this purpose, Chari et al. [11] perform the first formal security analysis of masking, by showing that the number of queries needed to recover a sensitive bit in a *noisy leakage model* is at least exponential in the masking order. In this model, the adversary gets leaked values sampled according to a Gaussian distribution centered around the actual value of the wire. This model is later extended by Prouff and Rivain [30] in several respects. First, they consider more general distributions to sample noisy leakage from, rather than just Gaussian [11] or Bernoulli leakage [18]. Moreover, they remove the limitation to one-bit observations, allowing the adversary to observe intermediate variables of any bitsize. Finally, they also extend the notion of leakage to take computation, rather than data, into account, following the *only*

computation leaks information principle introduced by Micali and Reyzin [24]. They also offer the first proof of security for a masked algorithm in this model, although it relies on leak-free components and a relatively weak adversary model.

In a different approach, Ishai, Sahai and Wagner [21] introduce the *t-threshold probing model*, in which the adversary receives the exact value of *at most t internal variables* (of his choice) in the computation. At the same time, they describe a transformation that turns any boolean circuit C secure in the *black-box model* into a circuit C' secure in the *t-threshold probing model*.

In practice, the noisy leakage model is often thought of as more realistic, since experimental physical leakage is noisy [23]. In particular, although the *t-threshold probing model* enables the adversary to observe exact values rather than noisy ones, it is not more powerful than the noisy leakage model, since the models also differ in the number of observations allowed to the adversary. The relationship between the two models was recently clarified by Duc, Dziembowski and Faust [14]. They advantageously recast the noisy leakage in the more classic statistical security model and show that security in the extended noisy leakage model of [30], fixed to capture chosen plaintext attacks, can be reduced to security in the *t-threshold probing model* of [21], in which security proofs are much more direct. In addition, the reduction does not rely on the existence of leak-free components. Thus, proving the security of a cryptosystem in the *t-threshold probing model* automatically ensures its security in the more realistic noisy leakage model.

In both models, only the values of intermediate variables are usually considered when determining the security order of an implementation. However, Balash et al. [2] show that this value-based leakage model does not fully capture some real-world scenarios in which additional physical leaks can occur, namely *glitches* or *transition-based* leakage, leaking information about more than one intermediate variable in a single observation. As a consequence, a perfectly masked algorithm secure in the value-based model can succumb to first-order attacks in these finer-grained leakage models.

Program Verification. Many tools aimed at proving the security of masked algorithms in the *t-threshold probing model* have recently appeared [9, 15–17, 25]. Some [9, 25] use type systems to propagate sensitivity marks along the programs, but such approaches are not complete [16] and many programs are thus incorrectly typed as secure. Others take the underlying probability distributions into account, but can only handle low masking orders (typically orders 1 and 2), even on small programs.

Contributions. In this paper, we develop automated methods to prove the security of masked implementations in the *t-threshold probing model*, both for value-based and transition-based leakage. More specifically, our theoretical contributions are three-fold: i. We provide a formal characterization of security in the *t-threshold probing model* as a combination of variants of two well-known properties in programming languages: *t*-non-interference and functional

equivalence; ii. We provide algorithms that construct bijections between an adversary observation and a distribution that is trivially independent from the secret inputs, thereby proving that the adversary observation is independent from secret inputs; iii. We provide algorithms that make use of the constructed bijections to extend sets of observations with additional observations that do not give the adversary any more information about the secrets, thereby reducing greatly the number of non-interference proofs that need to be performed in order to prove a whole program t -non-interfering. As a practical contribution, we implement our algorithms and apply them to various masked implementations of AES, and a masked implementation of MAC-Keccak. Pleasingly, our tools are able to successfully analyze first-order masked implementations of AES (in a couple of minutes), 2 rounds of second-order masked implementations of AES at level 2 (in around 22 minutes), and masked implementations of multiplication, up to order 5 (in 45s). Our experiments allow us to rediscover several known attacks ([12, 13]) on flawed implementations, to check that proposed fixes, when they exist, are indeed secure, and finally to discover new attacks on flawed implementations ([33]). We also discuss how our approach and tool can easily be adapted to deal with stronger leakage models capturing both transition-based leakage and leakage due to glitches, and illustrate it by studying the security of variants of secure field multiplication in the transition-based leakage model.

Putting our work in perspective, we deliberately focus on algorithmic methods that are able to cover large spaces of observation sets very efficiently, and without any assumption on the program. Although our results demonstrate that such methods can perform surprisingly well in practice, their inherent limitations with respect to scalability remain. A common strategy to address scalability issues is to develop compositional techniques. This could be done, for instance, in the context of a masking compiler, whose proof of security proceeds by showing that each gadget is secure, and that gadgets are combined securely. Assumptions on the structure of the masked algorithm could also be made that would allow such compositional reasoning. In this light, our algorithmic methods can be seen as focusing primarily on proving that core gadgets are secure with respect to a widely-used notion of security.

Outline. We first review previous uses of formal methods to prove similar properties (Section 2). In Sections 3 and 4, we describe our algorithms. In Section 5, we evaluate the practicality of our approach by implementing our algorithms in the framework provided by EasyCrypt [6], and testing the performance of our implementation on representative examples from the literature.

2 Language-Based Techniques for Threshold Security in the t -Threshold Probing Model

In this paper, we rephrase security in the t -threshold probing model by defining the notion of t -*non interference*, which is based on the notions of probabilistic non-interference used for verifying information-flow properties in language-based

security. We first define a general notion of program equivalence. Two probabilistic programs p_1 and p_2 are said to be $(\mathcal{I}, \mathcal{O})$ -equivalent, denoted $p_1 \sim_{\mathcal{I}}^{\mathcal{O}} p_2$, whenever the probability distributions on \mathcal{O} defined by p_1 and p_2 , conditioned by the assumptions on input variables encoded in \mathcal{I} , are equal.

This notion of equivalence subsumes the two more specialized notions we consider here: *functional equivalence* and *t-non-interference*. Two programs p and \bar{p} are said to be functionally equivalent when they are $(\mathcal{I}, \mathcal{Z})$ -equivalent with \mathcal{Z} all output variables, and \mathcal{I} all input variables. A program \bar{p} is said to be *t-non-interfering* with respect to a set of secret input variables \mathcal{I}_{sec} and a set of *observations* \mathcal{O} when $\bar{p}(s_0, \cdot)$ and $\bar{p}(s_1, \cdot)$ are $(\mathcal{I}_{\text{pub}}, \mathcal{O})$ -equivalent (with $\mathcal{I}_{\text{pub}} = \mathcal{I} \setminus \mathcal{I}_{\text{sec}}$ the set of non-secret input variables) for any values s_0 and s_1 of the secret input variables.

We now give an indistinguishability-based definition of the *t*-threshold probing model. In this model, the challenger randomly chooses two secret values s_0 and s_1 (representing for instance two different values of the secret key) and a bit b according to which the leakage will be produced: the output computation always uses secret s_0 , but the adversary observations are computed using s_b . The adversary \mathcal{A} is allowed to query an oracle with chosen instances of public arguments, along with a set of at most t intermediate variables (adaptively or non-adaptively chosen); such queries reveal their output and the values of the intermediate variables requested by the adversary. We say that \mathcal{A} wins if he guesses b .

We now state the central theorem to our approach, heavily inspired by Duc, Dziembowski and Faust [14] and Ishai, Sahai and Wagner [21].

Theorem 1. *Let p and \bar{p} be two programs. If p and \bar{p} are functionally equivalent and \bar{p} is t-non-interfering, then for every adversary \mathcal{A} against \bar{p} in the t-threshold probing model, there exists an adversary \mathcal{S} against p in the black-box model, such that*

$$\Delta(\mathcal{S} \xrightarrow{bb} p, \mathcal{A} \xrightarrow{thr} \bar{p}) = 0$$

where $\Delta(\cdot ; \cdot)$ denotes the statistical distance¹.

Proof. Since p and \bar{p} are functionally equivalent, we have $\Delta(\mathcal{S} \xrightarrow{bb} p, \mathcal{S} \xrightarrow{bb} \bar{p}) = 0$ for all black-box adversary \mathcal{S} , and we only have to prove that there exists an \mathcal{S} such that $\Delta(\mathcal{S} \xrightarrow{bb} \bar{p}, \mathcal{A} \xrightarrow{thr} \bar{p}) = 0$. We simply construct a simulator \mathcal{S}' that simulates the leakage for \mathcal{A} , and build \mathcal{S} by composing them. The simulator receives as inputs the public variables that are used for the execution of \bar{p} , and the output of \bar{p} , but not the t intermediate values corresponding to the observation set \mathcal{O} . Since \bar{p} is *t*-non-interfering, the observations do not depend on the secret variables that are used for the execution of \bar{p} , and the simulator can choose arbitrary values for the secret variables, run \bar{p} on these values and the public variables given as inputs, and output the requested observations. \square

¹ The theorem can be lifted to the noisy leakage model using Corollary 1 from [14], using a small bound on the statistical distance instead.

Following Theorem 1, we propose algorithms to prove functional equivalence (details can be found in the long version of this document [3]) and t -non-interference properties of probabilistic programs, thereby reducing the security of masked implementations in the t -threshold probing model to the black-box security of the algorithms they implement.

In the following, we provide an overview of language-based techniques that could be used to verify the assumptions of Theorem 1, and to motivate the need for more efficient techniques. First, we introduce mild variants of two standard problems in programming languages, namely information-flow checking and equivalence checking, which formalize the assumptions of Theorem 1. Then, we present three prominent methods to address these problems: type systems (which are only applicable to information-flow checking), model counting, and relational logics. Finally, we discuss efficiency issues and justify the need for efficient techniques.

2.1 Problem Statement and Setting

The hypotheses of Theorem 1 can be seen as variants of two problems that have been widely studied in the programming language setting: equivalence checking and information-flow checking. Equivalence checking is a standard problem in program verification, although it is generally considered in the setting of deterministic programs, whereas we consider probabilistic programs here. Information-flow checking is a standard problem in language-based security, although it usually considers flows from secret inputs to public outputs, whereas we consider flows from secret inputs to intermediate values here.

Both problems can be construed as instances of relational verification. For clarity, we formalize this view in the simple case of straightline probabilistic programs. Such programs are sequences of random assignments and deterministic assignments, and have distinguished sets of input and output variables. Given a program p , we let $\text{IVar}(p)$, $\text{OVar}(p)$, and $\text{PVar}(p)$ denote the sets of input, output, and intermediate variables of p . Without loss of generality, we assume that programs are written in single static assignment (SSA) form, and in particular, that program variables appear exactly once on the left hand side of an assignment, called their defining assignment—one can very easily transform an arbitrary straightline program into an equivalent straightline program in SSA form. Assuming that programs are in SSA form, we can partition $\text{PVar}(p)$ into two sets $\text{DVar}(p)$ and $\text{RVar}(p)$ of deterministic and probabilistic variables, where a variable is probabilistic if it is defined by a probabilistic assignment, and is deterministic otherwise. Let \mathcal{V} denote the set of program values (we ignore typing issues). Each program p can be interpreted as a function:

$$\llbracket p \rrbracket : \mathcal{D}(\mathcal{V}^\kappa) \rightarrow \mathcal{D}(\mathcal{V}^{\ell+\ell'})$$

where $\mathcal{D}(T)$ denotes the set of discrete distributions over a set T , and κ , ℓ and ℓ' respectively denote the sizes of $\text{IVar}(p)$, $\text{PVar}(p)$ and $\text{OVar}(p)$. The function $\llbracket p \rrbracket$ takes as input a joint distribution on input variables and returns a joint

distribution on all program variables, and is defined inductively in the expected way. Furthermore, one can define for every subset \mathcal{O} of $\text{PVar}(p)$ of size m a function:

$$\llbracket p \rrbracket_{\mathcal{O}} : \mathcal{D}(\mathcal{V}^{\kappa}) \rightarrow \mathcal{D}(\mathcal{V}^m)$$

that computes, for each $v \in \mathcal{V}^{\kappa}$, the marginal distributions of $\llbracket p \rrbracket(v)$ with respect to \mathcal{O} .

We can now define the information-flow checking problem formally: a program p is non-interfering with respect to a partial equivalence relation $\Phi \subseteq \mathcal{D}(\mathcal{V}^{\kappa}) \times \mathcal{D}(\mathcal{V}^{\kappa})$ (in the following, we write $\Phi \mu_1 \mu_2$ to mean $(\mu_1, \mu_2) \in \Phi$), and a set $\mathcal{O} \subseteq \text{PVar}(p)$, or (Φ, \mathcal{O}) -non-interfering, iff $\llbracket p \rrbracket_{\mathcal{O}}(\mu_1) = \llbracket p \rrbracket_{\mathcal{O}}(\mu_2)$ for every $\mu_1, \mu_2 \in \mathcal{D}(\mathcal{V}^{\kappa})$ such that $\Phi \mu_1 \mu_2$. In this case, we write $\text{NI}_{\Phi, \mathcal{O}}(p)$. Moreover, let \mathbb{O} be a set of subsets of $\text{PVar}(p)$, that is $\mathbb{O} \subseteq \mathcal{P}(\text{PVar}(p))$; we say that p is (Φ, \mathbb{O}) -non-interfering, if it is (Φ, \mathcal{O}) -non-interfering for every $\mathcal{O} \in \mathbb{O}$.

Before relating non-interference with security in the t -threshold probing models, we briefly comment on the nature of Φ . In the standard, deterministic, setting for non-interference, variables are generally marked as secret or public—in the general case, they can be drawn from a lattice of security levels, but this is not required here. Moreover, Φ denotes low equivalence, where two tuples of values v_1 and v_2 are low-equivalent if they coincide on public variables. The notion of low-equivalence has a direct counterpart in the probabilistic setting: two distributions μ_1 and μ_2 are low equivalent iff their marginal distributions with respect to public variables are equal. However, non-interference of masked implementations is often conditioned by well-formedness conditions on inputs; for instance, the inputs must consist of uniformly distributed, t -wise independent values. In this case, Φ is defined in such a way that two distributions are related by Φ iff they are well-formed and low equivalent.

There is a direct interpretation of t -threshold probing security as a non-interference property. We say that a program p is (Φ, t) -non-interfering if it is (Φ, \mathcal{O}) -non-interfering for all subsets \mathcal{O} of $\text{PVar}(p)$ with size smaller than t (we write $\mathcal{O} \in \mathcal{P}_{\leq t}(\text{PVar}(p))$ in the following). Then a program p is secure in the t -threshold probing model (with respect to a relation Φ) iff it is (Φ, t) -non-interfering.

In order to capture t -threshold probing security in the transition-based leakage model, we rely on a partial function `next` that maps program variables to their successors. For programs that have been translated into SSA form, all program variables are of the form x_i , where x is a variable of the original program, and i is an index—typically a program line number. The successor of such a variable x_i , when it exists, is a variable of the form x_j where j is the smallest index such that $i < j$ and x_j is a program variable. Then, we say that a program p is (Φ, t) -non-interfering in the transition-based model, written $\text{NI}_{\Phi, t, \text{succ}}(p)$, iff p is $(\Phi, \mathcal{O} \cup \text{next}(\mathcal{O}))$ -non-interfering for every subset of $\text{PVar}(p)$ with size smaller than t . Then a program p is secure in the transition-based t -threshold

probing model (with respect to a relation Φ) iff it is (Φ, t) -non-interfering in the transition-based model.²

We now turn to program equivalence. For the sake of simplicity, we consider two programs p_1 and p_2 that have the same sets of input and output variables; we let \mathcal{W} denote the latter. We let $\llbracket p \rrbracket_{\mathcal{W}}$ denote the function that computes for every initial distribution μ the marginal distribution of $\llbracket p \rrbracket(\mu)$ with respect to \mathcal{W} . We say that p_1 and p_2 are equivalent with respect to a partial equivalence relation $\Phi \subseteq \mathcal{D}(\mathcal{V}^\kappa) \times \mathcal{D}(\mathcal{V}^\kappa)$, written $p_1 \sim_\Phi p_2$, iff $\llbracket p_1 \rrbracket_{\mathcal{W}}(\mu) = \llbracket p_2 \rrbracket_{\mathcal{W}}(\mu)$ for every distribution μ such that $\Phi \mu \mu$.

For the sake of completeness, we point out that both notions are subsumed by the notion of (Φ, \mathcal{O}) -equivalence. Specifically, we say that programs p_1 and p_2 are (Φ, \mathcal{O}) -equivalent, written $p_1 \sim_{\Phi}^{\mathcal{O}} p_2$, iff $\llbracket p_1 \rrbracket_{\mathcal{O}}(\mu_1) = \llbracket p_2 \rrbracket_{\mathcal{O}}(\mu_2)$ for every two distributions μ_1 and μ_2 such that $\Phi \mu_1 \mu_2$. Therefore, both equivalence checking and information-flow checking can be implemented using as subroutine any sound algorithm for verifying that $p_1 \sim_{\Phi}^{\mathcal{O}} p_2$.

2.2 Type-Based Approaches

Information-flow type systems are a class of type systems that enforce non-interference by tracking dependencies between program variables and rejecting programs containing illicit flows. There are multiple notions of non-interference (termination-sensitive, termination-insensitive, or bisimulation-based) and forms of information-flow type systems (for instance, flow-sensitive, or flow-insensitive); we refer the reader to [32] for a survey. For the purpose of this paper, it is sufficient to know that information-flow type systems for deterministic programs assign to all program variables a level drawn from a lattice of security levels which includes a level of public variables and secret variables. In the same vein, one can develop information-flow type systems to enforce probabilistic non-interference; broadly speaking, such type systems distinguish between public values, secret values, and uniformly distributed values. Following these ideas, Moss et al. [25] pioneer the application of information-flow type systems to masking. They use the type system as a central part in a masking compiler that transforms an input program into a functionally equivalent program that is resistant to first-order DPA. Their technique can readily be extended to prove non-interference with respect to a single observation set.

Because they are implemented with well-understood tools (such as data flow analyses) and are able to handle large programs extremely fast, information-flow type systems provide an appealing solution that one would like to use for higher-order DPA. However, the semantic information carried by types is inherently attached to individual values, rather than tuples of values, and there is

² Similarly, glitches could be captured by considering that each observation leaks four values: the values of the arguments, and the old and new values of the wire or register. More fine-grained leakage models depending on implementation details and combining value-based, transition-based and glitch-based leakage could also be considered.

no immediately obvious way to devise an information-flow type system even for second-order DPA. Notwithstanding, it is relatively easy to devise a sound method for verifying resistance to higher-order DPA using an information-flow type system in the style of [25]. The basic idea is to instrument the code of the original program with assignments $w := x_1 \parallel \dots \parallel x_t$, where w is a fresh program variable, $x_1 \dots x_t$ are variables of the original program, and t is the order for which resistance is sought; we let p' denote the instrumented program. Clearly, a program p is secure at order t iff for every initial values \mathbf{v}_1 and \mathbf{v}_2 , $\llbracket p' \rrbracket_{\{w\}}(\mathbf{v}_1) = \llbracket p' \rrbracket_{\{w\}}(\mathbf{v}_2)$ where w ranges over the set of fresh variables that have been introduced by the transformation. It is then possible to use an information-flow type system in the spirit of [25] to verify that c' satisfies non-interference with respect to output set $\{w\}$. However, this transformational approach suffers from two shortcomings: first, a more elaborate type system is required for handling concatenation with sufficient accuracy; second, and more critically, the transformation induces an exponential blow-up in the size of programs.

In a slightly different context, Pettai and Laud [29] use a type-system to prove non-interference of a limited number of adversary observations imposed by their adversary model in the multi-party computation scenario. They do so by propagating information regarding linear dependencies on random variables throughout their arithmetic circuits and progressively replacing subcircuits with random gates. Because of the limited number of possible adversary observations their model imposes, they do not run into the same scalability issues we deal with in this paper. However, their techniques for dealing with active adversaries may be useful for verifying masking-based countermeasures in the presence of fault injection attacks.

2.3 SMT-Based Methods

There have been a number of works that use SMT solvers to achieve more flexible analysis of masked implementations.

Bayrak et al. [9] develop an SMT-based method for analyzing the sensitivity of sequences of operations. Informally, the notion of sensitivity characterizes whether a variable used to store an intermediate computation in the sequence of operations depends on a secret and is statistically independent from random variables. Their approach is specialized to first-order masking, and suffers from some scalability issue—in particular, they report analysis of a single round of AES.

Eldib, Wang and Schaumont develop an alternative tool, SCSniffer [16], that is able to analyze masked implementations at orders 1 and 2. Their approach is based on model counting [20]: to prove that a set of probabilistic expressions is distributed independently from a set of secrets, model-counting-based tools count the number of valuations of the secrets that yield each possible value of the observed expressions and checks that that number is indeed independent from the secret. This process in itself is inherently exponential in the size of the observed expressions, even when only one such observation is considered. To overcome this issue, SCSniffer implements an incremental approach for reducing

the size of such expressions when they contain randomness that is syntactically independent from the rest of the program. This incremental approach is essential to analyzing some of their examples, but it is still insufficient for analyzing complete implementations: for instance, SCSniffer can only analyze one round of (MAC-)Keccak whereas our approach is able to analyze the full 24 rounds of the permutation. The additional power of our tool is derived from our novel technique: instead of explicitly counting solutions to large boolean systems, our tool simply constructs a bijection between two distributions, one of which is syntactically independent from the secrets. Although the complexity of this process still depends on the size of expressions (and in particular in the number of randomly sampled variables they contain), it is only polynomial in it, rather than exponential. In addition, the approach, as it is used in Sleuth and SCSniffer, is limited to the study of boolean programs or circuits, where all variables are 1 bit in size. This leads to unwieldy program descriptions and artificially increases the size of expressions, thereby also artificially increasing the complexity of the problem. Our approach bypasses this issue by considering abstract algebraic expressions rather than specific types. This is only possible because we forego explicit solution counting. Moreover, SCSniffer requires to run the tool at all orders $d \leq t$ to obtain security at level t . In contrast, we achieve the same guarantees in a single run. This is due to the fact that the exclusive-or of observed variables is used for model counting rather than their joint distribution. Our approach yields proofs of t -non-interference directly by considering the joint distribution of observed variables. Finally, we contribute a technique that helps reduce the practical complexity of the problem by extending proofs of independence for a given observation set into a proof of independence for many observation sets at once. This process is made less costly by the fact that we can efficiently check whether a proof of independence is still valid for an extended observation set, but we believe it would apply to techniques based on model-counting given the same ability.

All of these differences lead to our techniques greatly outperforming existing approaches when it comes to practical examples. For example, even considering only masking at order 1, where it takes SCSniffer 10 minutes to prove a masked implementation of one round of Keccak (implemented bit-by-bit), it takes our tool around 7 minutes to prove the full 24 rounds of the permutation (implemented on 64-bit words as in reference implementations), and around 2 minutes to verify a full implementation of AES (including its key schedule).

2.4 Relational Verification

A more elaborate approach is to use program verification for proving non-interference and equivalence of programs. Because these properties are inherently relational—that is, they either consider two programs or two executions of the same program—the natural verification framework to establish such properties is relational program logic. Motivated by applications to cryptography, Barthe, Grégoire and Zanella-Béguelin [8] introduce pRHL, a probabilistic Relational Hoare Logic that is specifically tailored for the class of probabilistic programs

considered in this paper. Using pRHL, (ϕ, \mathcal{O}) -non-interference a program p is captured by the pRHL judgment:

$$\{\phi\}p \sim p \{ \bigwedge_{y \in \mathcal{O}} y\langle 1 \rangle = y\langle 2 \rangle \}$$

which informally states that the joint distributions of the variables $y \in \mathcal{O}$ coincide on any two executions (which is captured by the logical formula $y\langle 1 \rangle = y\langle 2 \rangle$) that start from initial memories related by Φ .

Barthe et al. [7] propose an automated method to verify the validity of such judgments. For clarity of our exposition, we consider the case where p is a straightline code program. The approach proceeds in three steps:

1. transform the program p into a semantically equivalent program which performs a sequence of random assignments, and then a sequence of deterministic assignments. The program transformation repeatedly applies eager sampling to pull all probabilistic assignments upfront. At this stage, the judgement is of the form

$$\{\phi\}S; D \sim S; D \{ \bigwedge_{y \in \mathcal{O}} y\langle 1 \rangle = y\langle 2 \rangle \}$$

where S is a sequence of probabilistic assignments, and D is a sequence of deterministic assignments;

2. apply a relational weakest precondition calculus to D the deterministic sequence of assignments; at this point, the judgment is of the form

$$\{\phi\}S \sim S \{ \bigwedge_{y \in \mathcal{O}} e_y\langle 1 \rangle = e_y\langle 2 \rangle \}$$

where e_y is an expression that depends only on the variables sampled in S and on the program inputs;

3. repeatedly apply the rule for random sampling to generate a verification condition that can be discharged by SMT solvers. Informally, the rule for random sampling requires finding a bijection between the domains of the distribution from which values are drawn, and proving that a formula derived from the post-condition is valid. We refer to [8] and [7] for a detailed explanation of the rule for random sampling. For our purposes, it is sufficient to consider a specialized logic for reasoning about the validity of judgments of the form above. We describe such a logic in Section 3.1.

Note that there is a mismatch between the definition of (Φ, t) -non-interference used to model security in the t -threshold probing model, and the notion of (ϕ, \mathcal{O}) -non-interference modelled by pRHL. In the former, Φ is a relation over distributions of memories, whereas in the latter ϕ is a relation over memories. There are two possible approaches to address this problem: the first is to develop a variant of pRHL that supports a richer language of assertions; while possible, the

resulting logic might not be amenable to automation. A more pragmatic solution, which we adopt in our tool, is to transform the program p into a program $i;p$, where i is some initialization step, such that p is (Φ, \mathcal{O}) non-interfering iff $i;p$ is (ϕ, \mathcal{O}) non-interfering for some pre-condition ϕ derived from Φ .

In particular, i includes code marked as non-observable that *preshares* any input or state marked as secret,³ and fully observable code that simply shares public inputs. The code for sharing and presharing, as well as a simple example of this transformation are given in Appendix A.

3 A Logic for Probabilistic Non-Interference

In this section, we propose new verification-based techniques to prove probabilistic non-interference statements. We first introduce a specialized logic to prove a vector of probabilistic expressions independent from some secret variables. We then explain how this logic specializes the general approach described in Section 2.4 to a particular interesting case. Finally, we describe simple algorithms that soundly construct derivations in our logic.

3.1 Our Logic

Our logic shares many similarities with the equational logic developed in [5] to reason about equality of distributions. In particular, it considers equational theories over multi-sorted signatures.

A multi-sorted signature is defined by a set of types and a set of operators. Each operator has a signature $\sigma_1 \times \dots \times \sigma_n \rightarrow \tau$, which determines the type of its arguments, and the type of the result. We assume that some operators are declared as invertible with respect to one or several of their arguments; informally, a k -ary operator f is invertible with respect to its i -th argument, or i -invertible for short, if, for any $(x_j)_{j \neq i}$ the function $f(x_0, \dots, x_{i-1}, \cdot, x_{i+1}, \dots, x_k)$ is a bijection. If f is i -invertible, we say that its i -th argument is an *invertible argument of f* .

Expressions are built inductively from two sets \mathcal{R} and \mathcal{X} of probabilistic and deterministic variables respectively, and from operators. Expressions are (strongly) typed. The set of deterministic (resp. probabilistic) variables of a vector of expressions e is denoted as $\text{dvar}(e)$ (resp. $\text{rvar}(e)$). We say that an expression e is *invertible in x* whenever $\forall i j, x \notin \text{rvar}(e_i^j)$, we have $e = f_1(\dots, e_{i_1-1}^1, f_2(\dots f_n(\dots, e_{i_n-1}^n, x, \dots) \dots), \dots)$, and each f_j is i_j -invertible.

We equip expressions with an equational theory \mathcal{E} . An equational theory is a set of equations, where an equation is a pair of expressions of the same type. Two expressions e and e' are provably equal with respect to an equational theory \mathcal{E} , written $e \doteq_{\mathcal{E}} e'$, if the equation $e \doteq_{\mathcal{E}} e'$ can be derived from the standard rules of multi-sorted equational logic: reflexivity, symmetry, transitivity, congruence, and instantiation of axioms in \mathcal{E} . Such axioms can be used, for example, to equip types with particular algebraic structures.

³ This corresponds to Ishai, Sahai and Wagner’s *input encoders* [21].

Expressions have a probabilistic semantics. A valuation ρ is a function that maps deterministic variables to values in the interpretation of their respective types. The interpretation $\llbracket e \rrbracket_\rho$ of an expression is a discrete distribution over the type of e ; informally, $\llbracket e \rrbracket_\rho$ samples all random variables in e , and returns the usual interpretation of e under an extended valuation ρ, ρ' where ρ' maps each probabilistic variable to a value of its type. The definition of interpretation is extended to tuples of expressions in the obvious way. Note that, contrary to the deterministic setting, the distribution $\llbracket (e_1, \dots, e_k) \rrbracket_\rho$ differs from the product distribution $\llbracket e_1 \rrbracket_\rho \times \dots \times \llbracket e_k \rrbracket_\rho$. We assume that the equational theory is consistent with respect to the interpretation of expressions.

Judgments in our logic are of the form $(\mathbf{x}_L, \mathbf{x}_H) \vdash e$, where e is a set of expressions and $(\mathbf{x}_L, \mathbf{x}_H)$ partitions the deterministic variables of e into public and private inputs, that is, $\text{dvar}(e) \subseteq \mathbf{x}_L \uplus \mathbf{x}_H$. A judgment $(\mathbf{x}_L, \mathbf{x}_H) \vdash e$ is valid iff the identity of distributions $\llbracket e \rrbracket_{\rho_1} = \llbracket e \rrbracket_{\rho_2}$ holds for all valuations ρ_1 and ρ_2 such that $\rho_1(x) = \rho_2(x)$ for all $x \in \mathbf{x}_L$.

A proof system for deriving valid judgments is given in Figure 1. Rule (INDEP) states that a judgment is valid whenever all the deterministic variables in expressions are public. Rule (CONV) states that one can replace expressions by other expressions that are provably equivalent with respect to the equational theory \mathcal{E} . Rule (OPT) states that, whenever the only occurrences of a random variable r in e are as the i -th argument of some fixed application of an i -invertible operator f where f 's other arguments are some $(e_j)_{j \neq i}$, then it is sufficient to derive the validity of the judgment where r is substituted for $f(e_0, \dots, e_{i-1}, r, e_{i+1}, \dots, e_k)$ in e . The soundness of rule (OPT) becomes clear by remarking that the distributions $\llbracket f(e_0, \dots, e_{i-1}, r, e_{i+1}, \dots, e_k) \rrbracket$ and $\llbracket r \rrbracket$ are equal, since f is i -invertible and r is uniform random and does not appear in any of the e_j . Although the proof system can be extended with further rules (see, for example [5]), these three rules are in fact sufficient for our purposes.

$$\begin{array}{c}
 \frac{\text{dvar}(e) \cap \mathbf{x}_H = \emptyset}{(\mathbf{x}_L, \mathbf{x}_H) \vdash e} \quad (\text{INDEP}) \qquad \frac{(\mathbf{x}_L, \mathbf{x}_H) \vdash e' \quad e \stackrel{\mathcal{E}}{=} e'}{(\mathbf{x}_L, \mathbf{x}_H) \vdash e} \quad (\text{CONV}) \\
 \frac{(\mathbf{x}_L, \mathbf{x}_H) \vdash e \quad f \text{ is } i\text{-invertible} \quad r \in \mathcal{R} \quad r \notin \text{rvar}(e_0, \dots, e_{i-1}, e_{i+1}, \dots, e_k)}{(\mathbf{x}_L, \mathbf{x}_H) \vdash e[f(e_0, \dots, e_{i-1}, r, e_{i+1}, \dots, e_k)/r]} \quad (\text{OPT})
 \end{array}$$

Fig. 1. Proof system for non-interference

3.2 From Logical Derivations to Relational Judgments

In Section 2.4, we have shown that the problem of proving that a program is (Φ, \mathcal{O}) -non-interfering could be reduced to proving relational judgements of the form $\{\phi\}S \sim S\{\bigwedge_{y \in \mathcal{O}} e_y \langle 1 \rangle = e_y \langle 2 \rangle\}$ where S is a sequence of random samplings, e_y is an expression that depends only on the variables sampled in S and on the program inputs, and ϕ is a precondition derived from Φ after

the initial sharing and presharing code is inserted, and exactly captures low-equivalence on the program’s inputs. We now show that proving such judgments can in fact be reduced to constructing a derivation in the logic from Section 3.1. Indeed, since both sides of the equalities in the postcondition are equal, it is in fact sufficient to prove that the $(e_y)_{y \in \mathcal{O}}$ are independent from secret inputs: since public inputs are known to be equal and both programs are identical, the postcondition then becomes trivially true. In particular, to prove the judgment $\{\bigwedge_{x \in \mathbf{x}_L} x\langle 1 \rangle = x\langle 2 \rangle\}S \sim S\{\bigwedge_{y \in \mathcal{O}} e_y\langle 1 \rangle = e_y\langle 2 \rangle\}$, it is in fact sufficient to find a derivation of $(\mathbf{x}_L, \mathbf{x}_H) \vdash (e_y)_{y \in \mathcal{O}}$, where \mathbf{x}_H is the complement of \mathbf{x}_L in the set of all program inputs. An example detailing this reasoning step is discussed in Appendix A.

3.3 Our Algorithms

We now describe two algorithms that soundly derive judgments in the logic. Throughout this paper, we make use of unspecified `choose` algorithms that, given a set X , return an $x \in X$ or \perp if $X = \emptyset$. We discuss our chosen instantiations where valuable.

Our simplest algorithm (Algorithm 1) works using only rules (INDEP) and (OPT) of the logic. Until (INDEP) applies, Algorithm 1 tries to apply (OPT), that is, to find (e', e, r) such that $r \in \mathcal{R}$ and e is invertible in r and $e = e'[e/r]$; if it succeeds, it then performs a recursive call on e' else it fails. Remark that the conditions are sufficient to derive the validity of e from the validity of e' using successive applications of the (OPT) rule.

The result of the function (\mathbf{h}) can be understood as a compact representation of the logical derivation. Such compact representations of derivations become especially useful in Section 4, where we efficiently extend sets of observed expressions, but can also be used, independently of performance, to construct formal proof trees if desired.

Algorithm 1. Proving Probabilistic Non-Interference: A Simple Algorithm

```

1: function  $\text{NI}_{\mathcal{R}, \mathbf{x}_H}(e)$             $\triangleright$  the joint distribution of  $e$  is independent from  $\mathbf{x}_H$ 
2:   if  $\forall x \in \text{dvar}(e). x \notin \mathbf{x}_H$  then
3:     return INDEP
4:    $(e', e, r) \leftarrow \text{choose}(\{(e', e, r) \mid e \text{ is invertible in } r \wedge r \in \mathcal{R} \wedge e = e'[e/r]\})$ 
5:   if  $(e', e, r) \neq \perp$  then
6:     return OPT( $e, r$ ) :  $\text{NI}_{\mathcal{R}, \mathbf{x}_H}(e')$ 
7:   return  $\perp$ 

```

This algorithm is sound, since it returns a derivation \mathbf{h} constructed after checking each rule’s side-conditions. However, it is incomplete and may fail to construct valid derivations. In particular, it does not make use of rule (CONV).

Our second algorithm (Algorithm 2) is a slight improvement on Algorithm 1 that makes restricted use of the (CONV) rule: when we cannot find a suitable (e', e, r) , we normalize algebraic expressions as described in [1], simplifying

expressions and perhaps revealing potential applications of the (OPT) rule. We use only algebraic normalization to avoid the need for user-provided hints, and even then, only use this restricted version of the (CONV) rule as a last resort. This is for two reasons: first, ring normalization may prevent the use of some (e', e, r) triples in later recursive calls (for example, the expression $(a + r) \cdot r'$ gets normalized as $a \cdot r' + r \cdot r'$, which prevents the substitution of $a + r$ by r); second, the normalization can be costly and negatively impact performance.

Algorithm 2. Proving Probabilistic Non-Interference: A More Precise Algorithm

```

1: function  $\text{NI}_{\mathcal{R}, \mathbf{x}_H}(e, b)$        $\triangleright$  the joint distribution of  $e$  is independent from  $\mathbf{x}_H$ 
2:   if  $\forall x \in \text{dvar}(e). x \notin \mathbf{x}_H$  then
3:     return INDEP
4:    $(e', e, r) \leftarrow \text{choose}(\{(e', e, r) \mid e \text{ is invertible in } r \wedge r \in \mathcal{R} \wedge e = e'[e/r]\})$ 
5:   if  $(e', e, r) \neq \perp$  then
6:     return OPT( $e, r$ ) :  $\text{NI}_{\mathcal{R}, \mathbf{x}_H}(e', b)$ 
7:   else if  $b$  then
8:      $e \leftarrow \text{ring\_simplify}(e)$ 
9:     return CONV :  $\text{NI}_{\mathcal{R}, \mathbf{x}_H}(e, \text{false})$ 
10:  return  $\perp$ 

```

In practice, we have found only one example where Algorithm 1 yields false negatives, and we have not found any where Algorithm 2 fails to prove the security of a secure implementation. In the following, we use $\text{NI}_{\mathcal{R}, \mathbf{x}_H}(X)$ the function from Algorithm 2 with b initially true. In particular, the implementation described and evaluated in Section 5 relies on this algorithm.⁴

Discussion. We observe that Algorithm 2 can only be refined in this way because it works directly on program expressions. In particular, any abstraction, be it type-based or otherwise, could prevent the equational theory from being used to simplify observed expressions. Further refinements are theoretically possible (in particular, we could also consider a complete proof system for the logic in Section 3.1), although they may be too costly to make use of in practice.

4 Divide-and-Conquer Algorithms Based on Large Sets

Even with efficient algorithms to prove that a program p is $(\mathcal{R}, \mathcal{O})$ -non-interfering for some observation set \mathcal{O} , proving that p is t -non-interfering remains a complex task: indeed this involves proving $\text{NI}_{\mathcal{R}, \mathcal{O}}(p)$ for all $\mathcal{O} \in \mathcal{P}_{\leq t}(\text{PVar}(p))$. Simply

⁴ Some of the longer-running experiments reported in Section 5 do make use of Algorithm 1 since their running time makes it impractical to run them repeatedly after algorithmic changes. However, Algorithm 2 only makes a difference when false positives occur, which is not the case on our long-running tests.

enumerating all possible observation sets quickly becomes intractable as p and t grow. Our main idea to solve this problem is based on the following fact: if $\text{NI}_{\mathcal{R}, \mathcal{O}}(p)$ then for every $\mathcal{O}' \subseteq \mathcal{O}$ we have $\text{NI}_{\mathcal{R}, \mathcal{O}'}(p)$. Therefore checking $\text{NI}_{\mathcal{R}, \mathcal{O}_i}(p)$ for every i can be done in a single step by checking $\text{NI}_{\mathcal{R}, \bigcup_i \mathcal{O}_i}(p)$.

Our goal is therefore to find fewer, larger observation sets $\mathcal{O}_1, \dots, \mathcal{O}_k$ such that $\text{NI}_{\mathcal{R}, \mathcal{O}_k}(p)$ for all k and, for all $\mathcal{O} \in \mathcal{P}_{\leq t}(\text{PVar}(\mathbf{p}))$, \mathcal{O} is a subset of at least one of the \mathcal{O}_i . Since this last condition is the contrapositive of the Hitting Set problem [19], which is known to be NP-hard, we do not expect to find a generally efficient solution, and focus on proposing algorithms that prove efficient in practice.

We describe and implement several algorithms based on the observation that the sequences of derivations constructed to prove the independence judgments in Section 2 can be used to efficiently extend the observation sets with additional observations whose joint distributions with the existing ones is still independent from the secrets. We first present algorithms that perform such extensions, and others that make use of observation sets extended in this way to find a family $\mathcal{O}_1, \dots, \mathcal{O}_k$ of observation sets that fulfill the condition above with k as small as possible.

4.1 Extending Safe Observation Sets

The $\text{NI}_{\mathcal{R}, \mathbf{x}_H}$ algorithm from Section 2 (Algorithm 2) allows us to identify sets X of expressions whose joint distribution is independent from variables in \mathbf{x}_H . We now want to extend such an X into a set X' that may contain more observable expressions and such that the joint distribution of X' is still independent from variables in \mathbf{x}_H .

First we define Algorithm 3, which rechecks that a derivation applies to a given set of expressions using the compact representation of derivations returned by algorithms 1 and 2: The algorithm simply checks that the consecutive rules encoded by \mathbf{h} can be applied on e . A key observation is that if $\text{NI}_{\mathcal{R}, \mathbf{x}_H}(e) = \mathbf{h}$ then $\text{recheck}_{\mathcal{R}, \mathbf{x}_H}(e, \mathbf{h})$. Furthermore, if $\text{recheck}_{\mathcal{R}, \mathbf{x}_H}(e, \mathbf{h})$ and $\text{recheck}_{\mathcal{R}, \mathbf{x}_H}(e', \mathbf{h})$ then $\text{recheck}_{\mathcal{R}, \mathbf{x}_H}(e \cup e', \mathbf{h})$.

Algorithm 3. Rechecking a derivation

```

function  $\text{recheck}_{\mathcal{R}, \mathbf{x}_H}(e, \mathbf{h})$   $\triangleright$  Check that the derivation represented by  $\mathbf{h}$  can be
    if  $\mathbf{h} = \text{INDEP}$  then
        return  $\forall x \in \text{dvar}(e). x \notin \mathbf{x}_H$ 
    if  $\mathbf{h} = \text{OPT}(e, r) : h'$  then
         $(e') \leftarrow \text{choose}(\{e' \mid e = e'[e/r]\})$ 
        if  $e' \neq \text{bot}$  then
            return  $\text{recheck}_{\mathcal{R}, \mathbf{x}_H}(e', h')$ 
    if  $\mathbf{h} = \text{CONV} : h'$  then
         $e \leftarrow \text{ring\_simplify}(e)$ 
        return  $\text{recheck}_{\mathcal{R}, \mathbf{x}_H}(e, h')$ 

```

Algorithm 4. Extending the Observation using a Fixed Derivation

```

function extend $\mathcal{R}, \mathbf{x}_H$ ( $\mathbf{x}, \mathbf{e}, \mathbf{h}$ )            $\triangleright$  find  $\mathbf{x}'$  such that  $\mathbf{x} \subseteq \mathbf{x}' \subseteq \mathbf{e}$  and  $\mathbf{h}(\mathbf{x}')$  is
     $\mathbf{e} \leftarrow \text{choose}(\mathbf{e})$           syntactically independent from  $\mathbf{x}_H$ 
    if recheck $\mathcal{R}, \mathbf{x}_H$ ( $\mathbf{e}, \mathbf{h}$ ) then
        return extend $\mathcal{R}, \mathbf{x}_H$ (( $\mathbf{x}, \mathbf{e}$ ),  $\mathbf{e} \setminus \{\mathbf{e}\}$ ,  $\mathbf{h}$ )
    else
        return extend $\mathcal{R}, \mathbf{x}_H$ ( $\mathbf{x}, \mathbf{e} \setminus \{\mathbf{e}\}$ ,  $\mathbf{h}$ )

```

Secondly, we consider (as Algorithm 4) an extension operation that only adds expressions on which \mathbf{h} can safely be applied as it is.

We also considered an algorithm that extends a set \mathbf{x} with elements in \mathbf{e} following \mathbf{h} whilst also extending the derivation itself when needed. However, this algorithm induces a loss of performance due to the low proportion of program variables that can in fact be used to extend the observation set, wasting a lot of effort on attempting to extend the derivation when it was not in fact possible. Coming up with a good choose algorithm that prioritizes variables that are likely to be successfully added to the observation set, and with conservative and efficient tests to avoid attempting to extend the derivation for variables that are clearly not independent from the secrets are interesting challenges that would refine this algorithm, and thus improve the performance of the space splitting algorithms we discuss next.

In the following, we use $\text{extend}_{\mathcal{R}, \mathbf{x}_H}(\mathbf{x}, \mathbf{e}, \mathbf{h})$ to denote the function from Algorithm 4, which is used to obtain all experimental results reported in Section 5.

4.2 Splitting the Space of Adversary Observations

Equipped with an efficient observation set extension algorithm, we can now attempt to accelerate the coverage of all possible sets of adversary observations to prove t -non-interference. The general idea of these coverage algorithms is to choose a set X of t observations and prove that the program is non-interfering with respect to X , then use the resulting derivation witness to efficiently extend X into an \widehat{X} that contains (hopefully many) more variables. This \widehat{X} , with respect to which the program is known to be non-interfering, can then be used to split the search space recursively. In this paper, we consider two splitting strategies to accelerate the enumeration: the first (Algorithm 5) simply splits the observation space into \widehat{X} and its complement before covering observations that straddle the two sets. The second (Algorithm 6) splits the space many-ways, considering all possible combinations of the sub-spaces when merging the sets resulting from recursive calls.

Pairwise Space-Splitting. Our first algorithm (Algorithm 5) uses its initial tuple X to split the space into two disjoint sets of observations, recursively descending into the one that does not supersede X and calling itself recursively to merge the two sets once they are processed separately.

Algorithm 5. Pairwise Space-Splitting

```

1: function  $\text{check}_{\mathcal{R}, \mathbf{x}_H}(\mathbf{x}, d, e)$   $\triangleright$  every  $\mathbf{x}, \mathbf{y}$  with  $\mathbf{y} \in \mathcal{P}_{\leq d}(e)$  is independent of  $\mathbf{x}_H$ 
2:   if  $d \leq |E|$  then
3:      $\mathbf{y} \leftarrow \text{choose}(\mathcal{P}_{\leq d}(e))$ 
4:      $\mathbf{h}_{\mathbf{x}, \mathbf{y}} \leftarrow \text{NI}_{\mathcal{R}, \mathbf{x}_H}((\mathbf{x}, \mathbf{y}))$   $\triangleright$  if  $\text{NI}_{\mathcal{R}, \mathbf{x}_H}$  fails, raise error  $\text{CannotProve}(\mathbf{x}, \mathbf{y})$ 
5:      $\widehat{\mathbf{y}} \leftarrow \text{extend}_{\mathcal{R}, \mathbf{x}_H}(\mathbf{y}, e \setminus \mathbf{y}, \mathbf{h}_{\mathbf{x}, \mathbf{y}})$   $\triangleright$  if  $\mathbf{h}_{\mathbf{x}, \mathbf{y}} = \top$ , use  $\widehat{\mathbf{y}} = \mathbf{y}$ 
6:      $\text{check}_{\mathcal{R}, \mathbf{x}_H}(\mathbf{x}, d, e \setminus \widehat{\mathbf{y}})$ 
7:     for  $0 < i < d$  do
8:       for  $\mathbf{u} \in \mathcal{P}_{\leq i}(\widehat{\mathbf{y}})$  do
9:          $\text{check}_{\mathcal{R}, \mathbf{x}_H}((\mathbf{x}, \mathbf{u}), d - i, e \setminus \widehat{\mathbf{y}})$ 

```

Theorem 2 (Soundness of Pairwise Space-Splitting). *Given a set \mathcal{R} of random variables, a set \mathbf{x}_H of secret variables, a set of expressions e and an integer $t > 0$, if $\text{check}_{\mathcal{R}, \mathbf{x}_H}(\emptyset, t, e)$ succeeds then every $\mathbf{x} \in \mathcal{P}_{\leq t}(e)$ is independent from \mathbf{x}_H .*

Proof. The proof is by generalizing on \mathbf{x} and d and by strong induction on e . If $|e| < d$, the theorem is vacuously true, and this base case is eventually reached since $\widehat{\mathbf{y}}$ contains at least d elements. Otherwise, by induction hypothesis, the algorithm is sound for every $e' \subsetneq e$. After line 5, we know that all t -tuples of variables in $\widehat{\mathbf{y}}$ are independent, jointly with \mathbf{x} , from the secrets. By the induction hypothesis, after line 6, we know that all t -tuples of variables in $e \setminus \widehat{\mathbf{y}}$ are independent, jointly with \mathbf{x} , from the secrets. It remains to prove the property for t -tuples that have some elements in $\widehat{\mathbf{y}}$ and some elements in $e \setminus \widehat{\mathbf{y}}$. The nested for loops at lines 7–9 guarantee it using the induction hypothesis. \square

Worklist-Based Space-Splitting. Our second algorithm (Algorithm 6) splits the space much more finely given an extended safe observation set. The algorithm works with a worklist of pairs (d, e) (initially called with a single element $(t, \mathcal{P}_{\leq t}(\text{PVar}(p)))$). Unless otherwise specified, we lift algorithms seen so far to work with vectors or sets of arguments by applying them element by element. Note in particular, that the for loop at line 7 iterates over all vectors of n integers such that each element i_j is strictly between 0 and d_j .

Algorithm 6. Worklist-Based Space-Splitting

```

1: function  $\text{check}_{\mathcal{R}, \mathbf{x}_H}((d_j, e_j)_{0 \leq j < n})$   $\triangleright$  every  $\mathbf{x} = \bigcup_{0 \leq j < n} \mathbf{x}_j$  with  $\mathbf{x}_j \in \mathcal{P}_{\leq d_j}(e_j)$  is independent from  $\mathbf{x}_H$ 
2:   if  $\forall j, d_j \leq |e_j|$  then
3:      $\mathbf{y}_j \leftarrow \text{choose}(\mathcal{P}_{\leq d_j}(e_j))$ 
4:      $\mathbf{h} \leftarrow \text{NI}_{\mathcal{R}, \mathbf{x}_H}(\bigcup_{0 \leq j < n} \mathbf{y}_j)$   $\triangleright$  if  $\text{NI}_{\mathcal{R}, \mathbf{x}_H}$  fails, raise error  $\text{CannotProve}(\bigcup \mathbf{y}_j)$ 
5:      $\widehat{\mathbf{y}}_j \leftarrow \text{extend}_{\mathcal{R}, \mathbf{x}_H}(\mathbf{y}_j, e_j \setminus \mathbf{y}_j, \mathbf{h})$ 
6:      $\text{check}_{\mathcal{R}, \mathbf{x}_H}((d_j, e_j \setminus \widehat{\mathbf{y}}_j)_{0 \leq j < n})$ 
7:     for  $j; 0 < i_j < d_j$  do
8:        $\text{check}_{\mathcal{R}, \mathbf{x}_H}(i_j, (\widehat{\mathbf{y}}_j, d_j - i_j, e_j \setminus \widehat{\mathbf{y}}_j))$ 

```

Theorem 3 (Soundness of Worklist-Based Space-Splitting). *Given a set \mathcal{R} of random variables, a set \mathbf{x}_H of secret variables, a set of expressions \mathbf{e} and an integer $t > 0$, if $\text{check}_{\mathcal{R}, \mathbf{x}_H}((t, \mathbf{e}))$ succeeds then every $\mathbf{x} \in \mathcal{P}_{\leq t}(\mathbf{e})$ is independent from \mathbf{x}_H .*

Proof. As in the proof of Theorem 2, we start by generalizing, and we prove that, for all vector (d_j, \mathbf{e}_j) with $0 < d_j$ for all j , if $\text{check}_{\mathcal{R}, \mathbf{x}_H}((d_j, \mathbf{e}_j))$ succeeds, then every $\mathbf{x} = \bigcup_{0 \leq j < n} \mathbf{x}_j$ with $\mathbf{x}_j \in \mathcal{P}_{\leq d_j}(\mathbf{e}_j)$ is independent from \mathbf{x}_H . The proof is again by strong induction on the vectors, using an element-wise lexicographic order (using size order on the \mathbf{e}) and lifting it to multisets as a bag order. If there exists an index i for which $|\mathbf{e}_i| < d_i$, the theorem is vacuously true. Otherwise, we unroll the algorithm in a manner similar to that in Theorem 2. After line 5, we know that, for every j , every $\mathbf{x} \in \mathcal{P}_{\leq d_j}(\widehat{\mathbf{y}}_j)$ is independent from \mathbf{x}_H . After line 6, by induction hypothesis (for all j , $\#\mathbf{e}_j \setminus \widehat{\mathbf{y}}_j < \#\mathbf{e}_j$ since $\widehat{\mathbf{y}}_j$ is of size at least d_j), we know that this is also the case for every $\mathbf{x} \in \mathcal{P}_{\leq d_j}(\widehat{\mathbf{y}}_j)$. Remains to prove that every subset of \mathbf{e}_j of size d_j that has some elements in $\widehat{\mathbf{y}}_j$ and some elements outside of it is also independent from \mathbf{x}_H . This is dealt with by the for loop on lines 7–8, which covers all possible combinations to recombine \mathbf{y}_j and its complement, in parallel for all j . \square

Comparison. Both algorithms lead to significant improvements in the verification time compared to the naive method which enumerates all t -tuples of observations for a given implementation. Further, our divide-and-conquer strategies make feasible the verification of some masked programs on which enumeration is simply unfeasible. To illustrate both these improvements and the differences between our algorithms, we apply the three methods to the S-box of [13] (Algorithm 4) protected at various orders. Table 1 shows the results, where column *# tuples* contains the total number of tuples of program points to be considered, column *# sets* contains the number of sets used by the splitting algorithms and the *time* column shows the verification times when run on a headless VM with a dual core⁵ 64-bit processor clocked at 2GHz.

As can be seen, the worklist-based method is generally the most efficient one. In the following, and in particular in Section 5, we use the `check` function from Algorithm 6.

Discussion. Note that in both Algorithms 5 and 6, the worst execution time occurs when the call to `extend` does not in fact increase the size of the observation set under study. In the unlikely event where this occurs in *all* recursive calls, both algorithms degrade into an exhaustive enumeration of all tuples, which is no worse than the naive implementation.

However, this observation makes it clear that it is important for the `extend` function to extend observation sets as much as possible. It could be interesting, and would definitely be valuable, to find a good balance between the complexity and precision of the `extend` function.

⁵ Only one core is used in the computation.

Table 1. Comparison of Algorithms 5 and 6 with naive enumeration and with each other

Method	# tuples	Security	Complexity	
			# sets	time
First-Order Masking				
naive	63	✓	63	0.001s
pair	63	✓	17	0.001s
list	63	✓	17	0.001s
Second-Order Masking				
naive	12,561	✓	12,561	0.180s
pair	12,561	✓	851	0.046s
list	12,561	✓	619	0.029s
Third-Order Masking				
naive	4,499,950	✓	4,499,950	140.642s
pair	4,499,950	✓	68,492	9.923s
list	4,499,950	✓	33,075	3.894s
Fourth-Order Masking				
naive	2,277,036,685	✓	-	unpractical
pair	2,277,036,685	✓	8,852,144	2959.770s
list	2,277,036,685	✓	3,343,587	879.235s

5 Experiments

In this section, we aim to show on concrete examples the efficiency of the methods we considered so far. This evaluation is performed using a prototype implementation of our algorithms that uses the EasyCrypt [6] tool’s internal representations of programs and expressions, and relying on some of its low-level tactics for substitution and conversion. As such, the prototype is not designed for performance, but rather for trust, and the time measurements given below could certainly be improved. However, the numbers of sets each algorithm considers are fixed by our choice of algorithm, and by the particular `choose` algorithms we decided to use. We detail and discuss this particular implementation decision at the end of this section.

Our choice of examples mainly focuses on higher-order masking schemes since they are much more promising than the schemes dedicated to small orders. Aside from the masking order itself, the most salient limiting factor for performance is the size of the program considered, which is also (more or less) the number of observations that need to be considered. Still, we analyze programs of sizes ranging from simple multiplication algorithms to either round-reduced or full AES, depending on the masking order.

We discuss our practical results depending on the leakage model considered: we first discuss our prototype’s performance in the value-based leakage model, then focus on results obtained in the transition-based leakage model.

Table 2. Verification of state-of-the-art higher-order masking schemes with # tuples the number t -uples of the algorithm at order t , # sets the number of sets built by our prototype and time the verification time in seconds

Reference	Target	# tuples	Result	Complexity # sets	time (s)
First-Order Masking					
CHES10 [31]	multiplication	13	secure ✓	7	ε
FSE13 [13]	Sbox (4)	63	secure ✓	17	ε
FSE13 [13]	full AES (4)	17,206	secure ✓	3,342	128
MAC-Keccak	full Keccak-f	13,466	secure ✓	5,421	405
Second-Order Masking					
RSA06 [33]	Sbox	1,188,111	secure ✓	4,104	1.649
CHES10 [31]	multiplication	435	secure ✓	92	0.001
CHES10 [31]	Sbox	7,140	1 st -order flaws (2)	866	0.045
CHES10 [31]	key schedule [13]	23,041,866	secure ✓	771,263	340,745
FSE13 [13]	AES 2 rounds (4)	25,429,146	secure ✓	511,865	1,295
FSE13 [13]	AES 4 rounds (4)	109,571,806	secure ✓	2,317,593	40,169
Third-Order Masking					
CHES10 [31]	multiplication	24,804	secure ✓	1,410	0.033
FSE13 [13]	Sbox(4)	4,499,950	secure ✓	33,075	3.894
FSE13 [13]	Sbox(5)	4,499,950	secure ✓	39,613	5.036
Fourth-Order Masking					
RSA06 [33]	Sbox	4,874,429,560	3 rd -order flaws (98, 176)	35,895,437	22,119
CHES10 [31]	multiplication	2,024,785	secure ✓	33,322	1.138
FSE13 [13]	Sbox (4)	2,277,036,685	secure ✓	3,343,587	879
Fifth-Order Masking					
CHES10 [31]	multiplication	216,071,394	secure ✓	856,147	45

5.1 Value-Based Model

Table 2 lists the performance of our prototype on multiple examples, presenting the total number of sets of observations to be considered (giving an indication of each problem’s relative difficulty), as well as the number of sets used to cover all tuples of observations by our prototype. We also list the verification time, although these could certainly be improved independently of the algorithms themselves. Each of our tests is identified by a reference and a function, with additional information where relevant. The MAC-Keccak example is a simple implementation of Keccak-f on 64-bit words, masked using a variant of Ishai, Sahai and Wagner’s transformation [21, 31] (noting that their SecMult algorithm can be used to securely compute any associative and commutative binary operation that distributes over field addition, including bitwise ANDs).

The two rows without checkmarks correspond to examples on which the tool fails to prove t -non-interference. We now analyze them in more detail.

On Schramm and Paar’s table-based implementation of the AES Sbox, supposed to be secure at order 4, our tool finds 98,176 third-order observations that

it cannot prove independent from the secrets. The time listed is the time needed to cover all triples, and the first error is found in 0.221s. These errors in fact correspond to four families of observations, which we now describe. Denoting by $X = \bigoplus_{0 \leq i \leq 4} x_i$ the S-box input and by $Y = \bigoplus_{0 \leq i \leq 4} y_i$ its output, we can write the four sets of flawed triples as follows:

1. $(x_0, \text{Sbox}(X \oplus x_0 \oplus i) \oplus (Y \oplus y_0), \text{Sbox}(X \oplus x_0 \oplus j) \oplus (Y \oplus y_0))$,
 $\forall i, j \in \text{GF}(2^8), i \neq j$
2. $(y_0, \text{Sbox}(X \oplus x_0 \oplus i) \oplus (Y \oplus y_0), \text{Sbox}(X \oplus x_0 \oplus j) \oplus (Y \oplus y_0))$,
 $\forall i, j \in \text{GF}(2^8), i \neq j$
3. $(x_0, \text{Sbox}(X \oplus x_0 \oplus i) \oplus (Y \oplus y_0 \oplus y_4), \text{Sbox}(X \oplus x_0 \oplus j) \oplus (Y \oplus y_0 \oplus y_4))$,
 $\forall i, j \in \text{GF}(2^8), i \neq j$
4. $(x_0, y_0, \text{Sbox}(X \oplus x_0 \oplus i) \oplus (Y \oplus y_0))$, $\forall i \in \text{GF}(2^8)$.

We recall that y_0 is read as $y_0 = \text{Sbox}(x_0)$, and prove that all four families of observations in fact correspond to attacks.

The first family corresponds to the attack detailed by Coron, Prouff and Rivain [12]). By summing the second and third variables, the attacker obtains $\text{Sbox}(X \oplus x_0 \oplus i) \oplus \text{Sbox}(X \oplus x_0 \oplus j)$. The additional knowledge of x_0 clearly breaks the independence from X . To recover secrets from a second set's triple of observations, the attacker can sum the second and third variables to obtain $X \oplus x_0$, from which he can learn $Y \oplus y_0$ (by combining it with the second variable) and then Y (by combining it with the first one). The third family is a variant of the first: the S-box masks can be removed in both cases. Finally, when observing three variables in the fourth family of observations, the knowledge of both x_0 and y_0 unmasks the third observed variable, making it dependent on X .

Our tool also finds two suspicious adversary observations on the S-box algorithm proposed by Rivain and Prouff [31], that in fact correspond to the two flaws revealed in [13]. However, by the soundness of our algorithm, and since our implementation only reports these two flaws, we now know that these are the only two observations that reveal any information on the secrets. We consider several corrected versions of this S-box algorithm, listed in Table 3. Some of these fixes focused on using a more secure mask refreshing function (borrowed from [14]) or refreshing all modified variables that are reused later on (as suggested by [30]). Others make use of specialized versions of the multiplication algorithm [13] that allow the masked program to retain its performance whilst gaining in security.

Although it is important to note that the algorithms appear to be “precise enough” in practice, Table 2 also reveals that program size is not in fact the only source of complexity. Indeed, proving the full key schedule at order 2 only involves around 23 million pairs of observations, compared to the 109 million that need to be considered to prove the security of 4 rounds of AES at the same order; yet the latter takes less than an hour to complete compared to 4 days for the full ten rounds of key schedule. We suspect that this is due to the shapes of the two programs’ dependency graphs, with each variable in the key schedule depending on a large proportion of the program’s input variables, whereas the dependencies in full AES are sparser. Although properties of composition would

Table 3. Fixing RP-CHE10 [31] at the second order

Reference	S-box	# tuples	Result	Complexity # sets time
Second-Order Masking				
RP-CHE10 [31]	initially proposed	7,140	1 st -order flaws (2)	840 0.070s
RP-CHE10 [31]	different refreshMasks	7,875	secure ✓	949 0.164s
RP-CHE10 [31]	more refreshMasks	8,646	secure ✓	902 0.180s
CPRR-FSE13 [13]	use of $x \cdot g(x)$ (Algo 4)	12,561	secure ✓	619 0.073s
CPRR-FSE13 [13]	use of tables (Algo 5)	12,561	secure ✓	955 0.196s

allow us to consider large programs masked at much higher orders, we leave these investigations to further works.

Another important factor in the performance of our algorithm is the instantiation of the various choice functions. We describe them here for the sake of reproducibility. In Algorithms 1 and 2, when choosing a triple (\mathbf{e}', e, r) to use with rule (OPT), our prototype first chooses r as the first (leftmost-first depth-first) random variable that fulfills the required conditions, then chooses e as the *largest* superterm of r that fulfills the required conditions (this fixes \mathbf{e}'). When choosing an expression to observe (in Algorithms 5 and 6) or to extend a set of observation with (in Algorithm 4), we choose first the expression that has the highest number of dependencies on random or input variables. These decisions certainly may have a significant effect on our algorithm’s performance, and investigating these effects more deeply may help gather some insight on the core problems related to masking. We leave this a future work.

5.2 Transition-Based Model

The value-based leakage model may not always be the best fit to capture the behaviour of hardware and software. In particular, when considering software implementations, it is possible that writing a value into a register leaks both its new and old contents. To illustrate the adaptability of our algorithms, we first run some simple tests. We then illustrate another potential application of our tool, whereby masked implementations that make use of $t+1$ masks per variable can be proved secure in the transitions model at orders much higher than the generic $t/2$, simply by reordering instructions and reallocating registers.

Table 4 describes the result of our experiments. Our first (naive) implementation is only secure at the second order in the transition-based leakage model and uses 21 local registers (the number of registers needed for this and other implementations to be secure could also be reduced further by zeroing out registers between independent uses). Our first improved implementation achieves security at order 3 in the transition-based leakage model with only 6 local registers. Trying to provide the best possible security in this model, we also find a third implementation that achieves security at order 4. This last implementation is in fact the original implementation with additional registers. Note however,

Table 4. Multiplication in the transition-based leakage model

Reference	Multiplication	# tuples	Security	Complexity # sets	Complexity time
RP-CHE10 [31]	initial scheme for order 4	3,570	order 2	161	0.008s
RP-CHE10 [31]	with some instructions reordering	98,770	order 3	3,488	0.179s
RP-CHE10 [31]	using more registers	2,024,785	order 4	17,319	1.235s

that in spite of its maximal security order, this last implementation still reuses registers (in fact, most are used at least twice).

The main point of these experiments is to show that the techniques and tools we developed are helpful in building and verifying implementations in other models. Concretely, our tools give countermeasure designers the chance to easily check the security of their implementation in one or the other leakage model, and identify problematic observations that would prevent the countermeasure from operating properly against higher-order adversaries.

6 Conclusion

This paper initiates the study of relational verification techniques for checking the security of masked implementations against t -order DPA attacks. Beyond demonstrating the feasibility of this approach for masking orders higher than 2, our work opens a number of interesting perspectives on automated DPA tools.

The most immediate direction for further work is to exhibit and prove compositional properties in order to achieve the verification of larger masked programs at higher orders.

Another promising direction is to automatically synthesize efficient and secure implementations by search-based optimization. Specifically, we envision a 2-step approach where one first uses an unoptimized but provably secure compiler to transform a program p into a program \bar{p}_t that is t -non-interfering, and then applies relational synthesis methods, in the spirit of [4], to derive a more efficient program p' that is observationally equivalent to \bar{p}_t and equally secure—the latter property being verified using PRHL.

Acknowledgments. We thank F.-X. Standaert and V. Grossi for giving us access to their examples and code generation tools, and H. Eldib and C. Wang for letting us make use of their code samples for comparison purposes. This research is partially funded by Spanish projects TIN2009-14599 DESAFIOS 10 and TIN2012-39391-C04-01 StrongSoft, Madrid Regional project S2009TIC-1465 PROMETIDOS, ANR project ANR-14-CE28-0015 BRUTUS and ANR-10-SEGI-015 PRINCE.

A Initial Transformations on Programs: An Example

To illustrate our algorithms, we consider the simple masked multiplication algorithm defined in [31] and relying on Algorithm 7, which is secure against 2-threshold probing adversaries. In practice, the code we consider is in 3-address

form, with a single operation per line (operator application or table lookup). For brevity, we use parentheses instead, unless relevant to the discussion. In the rest of this paper, we write Line $(n).i$ to denote the i^{th} expression computed on line n , using the convention that products are computed immediately before their use. For example, Line (5).1 is the expression $a_0 \odot b_1$, Line (5).2 is $r_{0,1} \oplus a_0 \odot b_1$ and Line (5).3 is $a_1 \odot b_0$.

Algorithm 7. Secure Multiplication Algorithm ($t = 2$) from [31]

Input: a_0, a_1, a_2 (resp. b_0, b_1, b_2) such that $a_0 \oplus a_1 \oplus a_2 = a$ (resp. $b_0 \oplus b_1 \oplus b_2 = b$)

Output: c_0, c_1, c_2 such that $c_0 \oplus c_1 \oplus c_2 = a \odot b$

```

1: function SECMULT( $\llbracket a_0, a_1, a_2 \rrbracket, \llbracket b_0, b_1, b_2 \rrbracket$ )
2:    $r_{0,1} \xleftarrow{\$} \mathbb{F}_{256}$ 
3:    $r_{0,2} \xleftarrow{\$} \mathbb{F}_{256}$ 
4:    $r_{1,2} \xleftarrow{\$} \mathbb{F}_{256}$ 
5:    $r_{1,0} \leftarrow (r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0$ 
6:    $r_{2,0} \leftarrow (r_{0,2} \oplus a_0 \odot b_2) \oplus a_2 \odot b_0$ 
7:    $r_{2,1} \leftarrow (r_{1,2} \oplus a_1 \odot b_2) \oplus a_2 \odot b_1$ 
8:    $c_0 \leftarrow (a_0 \odot b_0 \oplus r_{0,1}) \oplus r_{0,2}$ 
9:    $c_1 \leftarrow (a_1 \odot b_1 \oplus r_{1,0}) \oplus r_{1,2}$ 
10:   $c_2 \leftarrow (a_2 \odot b_2 \oplus r_{2,0}) \oplus r_{2,1}$ 
11:  return  $\llbracket c_0, c_1, c_2 \rrbracket$ 

```

Algorithm 8. Presharing, Sharing and Preprocessed multiplication ($t = 2$, a is secret, b is public)

<pre> 1: function PRESHARE(a) 2: $a_0 \xleftarrow{\\$} \mathbb{F}_{256}$ 3: $a_1 \xleftarrow{\\$} \mathbb{F}_{256}$ 4: $a_2 \leftarrow [a \oplus a_0 \oplus a_1]$ 5: return $\llbracket a_0, a_1, a_2 \rrbracket$ </pre>	<pre> 1: function SECMULT(a, b) 2: $a_0 \xleftarrow{\\$} \mathbb{F}_{256}$ 3: $a_1 \xleftarrow{\\$} \mathbb{F}_{256}$ 4: $a_2 \leftarrow [a \oplus a_0 \oplus a_1]$ 5: $b_0 \xleftarrow{\\$} \mathbb{F}_{256}$ 6: $b_1 \xleftarrow{\\$} \mathbb{F}_{256}$ 7: $b_2 \leftarrow (b \oplus b_0) \oplus b_1$ 8: $r_{0,1} \xleftarrow{\\$} \mathbb{F}_{256}$ 9: $r_{0,2} \xleftarrow{\\$} \mathbb{F}_{256}$ 10: $r_{1,2} \xleftarrow{\\$} \mathbb{F}_{256}$ 11: $r_{1,0} \leftarrow (r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0$ 12: $r_{2,0} \leftarrow (r_{0,2} \oplus a_0 \odot b_2) \oplus a_2 \odot b_0$ 13: $r_{2,1} \leftarrow (r_{1,2} \oplus a_1 \odot b_2) \oplus a_2 \odot b_1$ 14: $c_0 \leftarrow (a_0 \odot b_0 \oplus r_{0,1}) \oplus r_{0,2}$ 15: $c_1 \leftarrow (a_1 \odot b_1 \oplus r_{1,0}) \oplus r_{1,2}$ 16: $c_2 \leftarrow (a_2 \odot b_2 \oplus r_{2,0}) \oplus r_{2,1}$ 17: return $[c_0 \oplus c_1 \oplus c_2]$ </pre>
---	--

Line	Observed Expression	Line	Observed Expression
(2)	a_0	(12).2	$r_{0,2} \oplus a_0 \odot b_2$
(3)	a_1	(12).3	$a_2 \odot b_0$
(4)	$a_2 := (a \oplus a_0) \oplus a_1$	(12)	$(r_{0,2} \oplus a_0 \odot b_2) \oplus a_2 \odot b_0$
(5)	b_0	(13).1	$a_1 \odot b_2$
(6)	b_1	(13).2	$r_{1,2} \oplus a_1 \odot b_2$
(7).1	$b \oplus b_0$	(13).3	$a_2 \odot b_1$
(7)	$b_2 := (b \oplus b_0) \oplus b_1$	(13)	$(r_{1,2} \oplus a_1 \odot b_2) \oplus a_2 \odot b_1$
(8)	$r_{0,1}$	(14).1	$a_0 \odot b_0$
(9)	$r_{0,2}$	(14).2	$a_0 \odot b_0 \oplus r_{0,1}$
(10)	$r_{1,2}$	(14)	$(a_0 \odot b_0 \oplus r_{0,1}) \oplus r_{0,2}$
(11).1	$a_0 \odot b_1$	(15).1	$a_1 \odot b_1$
(11).2	$r_{0,1} \oplus a_0 \odot b_1$	(15).2	$a_1 \odot b_1 \oplus ((r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0)$
(11).3	$a_1 \odot b_0$	(15)	$(a_1 \odot b_1 \oplus ((r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0)) \oplus r_{1,2}$
(11)	$(r_{0,1} \oplus a_0 \odot b_1) \oplus a_1 \odot b_0$	(16).1	$a_2 \odot b_2$
(12).1	$a_0 \odot b_2$	(16).2	$a_2 \odot b_2 \oplus ((r_{0,2} \oplus a_0 \odot b_2) \oplus a_2 \odot b_0)$
		(16)	$(16).2 \oplus ((r_{1,2} \oplus a_1 \odot b_2) \oplus a_2 \odot b_1)$

Fig. 2. Possible wire observations for $\overline{\text{SECMULT}}$. (Note that, after Lines 4 and 7, we keep a_2 and b_2 in expressions due to margin constraints.)

When given a program whose inputs have been annotated as secret or public, we transform it as described at the end of Section 2.4 to add some simple initialization code that preshares secrets in a way that is not observable by the adversary, and lets the adversary observe the initial sharing of public inputs. This allows us to model, as part of the program, the assumption that shares of the secret are initially uniformly distributed and that their sum is the secret. The initialization code, as well as the transformed version of Algorithm 7 where argument a is marked as secret and b is marked as public, are shown in Algorithm 8. We use the square brackets on Line (4) of function PRESHARE to mean that the intermediate results obtained during the computation of the bracketed expression are not observable by the adversary: this is equivalent to the usual assumption that secret inputs and state are shared before the adversary starts performing measurements.

Once the program is in this form, it can be transformed to obtain: i. the set of its random variables;⁶ ii. the set of expressions representing all of the possible adversary observations; This final processing step on SECMULT yields the set of random variables $\mathcal{R} = \{a_0, a_1, b_0, b_1, r_{0,1}, r_{0,2}, r_{1,2}\}$, and the set of expressions shown in Figure 2 (labelled with their extended line number). Recall that these sets were obtained with a marked as secret and b marked as public.

⁶ In practice, since we consider programs in SSA form, it is not possible to assign a non-random value to a variable that was initialized with a random.

Line	Register	Old Contents	New Contents
(2)		\perp	a_0
(3)		\perp	a_1
(4)		\perp	$a \oplus a_0 \oplus a_1$
(5)		\perp	b_0
(6)		\perp	b_1
(7).1	b_2	\perp	$b \oplus b_0$
(7)		$b \oplus b_0$	$b \oplus b_0 \oplus a_1$
(8)		\perp	$r_{0,1}$
(9)		\perp	$r_{0,2}$
(10)		\perp	$r_{1,2}$
(11).1	$r_{1,0}$	\perp	$a_0 \odot b_1$
(11).2	$r_{1,0}$	$a_0 \odot b_1$	$r_{0,1} \oplus a_0 \odot b_1$
(11).3	t	\perp	$a_1 \odot b_0$
(11)		$r_{0,1} \oplus a_0 \odot b_1$	$r_{0,1} \oplus a_0 \odot b_1 \oplus a_1 \odot b_0$
(12).1	$r_{2,0}$	\perp	$a_0 \odot b_2$
(12).2	$r_{2,0}$	$a_0 \odot b_2$	$r_{0,2} \oplus a_0 \odot b_2$
(12).3	t	$a_1 \odot b_0$	$a_2 \odot b_0$
(12)		$r_{0,2} \oplus a_0 \odot b_2$	$r_{0,2} \oplus a_0 \odot b_2 \oplus a_2 \odot b_0$
(13).1	$r_{2,1}$	\perp	$a_1 \odot b_2$
(13).2	$r_{2,1}$	$a_1 \odot b_2$	$r_{1,2} \oplus a_1 \odot b_2$
(13).3	t	$a_2 \odot b_0$	$a_2 \odot b_1$
(13)		$r_{1,2} \oplus a_1 \odot b_2$	$r_{1,2} \oplus a_1 \odot b_2 \oplus a_2 \odot b_1$
(14).1	c_0	\perp	$a_0 \odot b_0$
(14).2	c_0	$a_0 \odot b_0$	$a_0 \odot b_0 \oplus r_{0,1}$
(14)		$a_0 \odot b_0 \oplus r_{0,1}$	$a_0 \odot b_0 \oplus r_{0,1} \oplus r_{0,2}$
(15).1	c_1	\perp	$a_1 \odot b_1$
(15).2	c_1	$a_1 \odot b_1$	$a_1 \odot b_1 \oplus r_{0,1} \oplus a_0 \odot b_1 \oplus a_1 \odot b_0$
(15)		$a_1 \odot b_1 \oplus r_{0,1} \oplus a_0 \odot b_1 \oplus a_1 \odot b_0$	$(15).2 \oplus r_{1,2}$
(16).1	c_2	\perp	$a_2 \odot b_2$
(16).2	c_2	$a_2 \odot b_2$	$a_2 \odot b_2 \oplus r_{0,2} \oplus a_0 \odot b_2 \oplus a_2 \odot b_0$
(16)		$a_2 \odot b_2 \oplus r_{0,2} \oplus a_0 \odot b_2 \oplus a_2 \odot b_0$	$(16).2 \oplus r_{1,2} \oplus a_1 \odot b_2 \oplus a_2 \odot b_1$

Fig. 3. Possible transition observations for $\overline{\text{SECMULT}}$ with a naive register allocation (shown in the last column). \perp denotes an uninitialized register, whose content may already be known to (and perhaps chosen by) the adversary.

A.1 Observable Transitions

Figure 3 presents the observable transitions for Algorithm 7. It gives the old value and the new value of the register modified by each program point. This is done using a simple register allocation of Algorithm 7 (where we use the word “register” loosely, to denote program variables, plus perhaps some additional temporary registers if required) that uses a single temporary register that is never cleared, and stores intermediate computations in the variable where their end result is stored. For clarity, the register in which the intermediate result is stored is also listed in the Figure.

References

1. Akinyele, J., Barthe, G., Grégoire, B., Schmidt, B., Strub, P.-Y.: Certified synthesis of efficient batch verifiers. In: 27th IEEE Computer Security Foundations Symposium, CSF 2014. IEEE Computer Society (2014) (to appear)
2. Balasch, J., Gierlich, B., Gross, V., Reparaz, O., Standaert, F.-X.: On the cost of lazy engineering for masked software implementations. Cryptology ePrint Archive, Report 2014/413 (2014). <http://eprint.iacr.org/2014/413>
3. Barthe, G., Belaid, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y.: Verified proofs of higher-order masking. Cryptology ePrint Archive, Report 2015/060 (2015). <http://eprint.iacr.org/>
4. Barthe, G., Crespo, J.M., Gulwani, S., Kunz, C., Marron, M.: From relational verification to SIMD loop synthesis. In: Nicolau, A., Shen, X., Amarasinghe, S.P., Vuduc, R.W. (eds.) Principles and Practice of Parallel Programming (PPoPP), pp. 123–134. ACM (2013)
5. Barthe, G., Daubignard, M., Kapron, B., Lakhnech, Y., Laporte, V.: On the equality of probabilistic terms. In: Clarke, E.M., Voronkov, A. (eds.) LPAR-16. LNCS, vol. 6355, pp. 46–63. Springer, Heidelberg (2010)
6. Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., Strub, P.-Y.: EasyCrypt: a tutorial. In: Aldini, A., Lopez, J., Martinelli, F. (eds.) FOSAD VII. LNCS, vol. 8604, pp. 146–166. Springer, Heidelberg (2014)
7. Barthe, G., Grégoire, B., Heraud, S., Zanella-Béguelin, S.: Computer-aided security proofs for the working cryptographer. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 71–90. Springer, Heidelberg (2011)
8. Barthe, G., Grégoire, B., Zanella-Béguelin, S.: Formal certification of code-based cryptographic proofs. In: 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, pp. 90–101. ACM (2009)
9. Bayrak, A.G., Regazzoni, F., Novo, D., Ienne, P.: Sleuth: automated verification of software power analysis countermeasures. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 293–310. Springer, Heidelberg (2013)
10. Canright, D., Batina, L.: A very compact “perfectly masked” S-box for AES. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 446–459. Springer, Heidelberg (2008)
11. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)
12. Coron, J.-S., Prouff, E., Rivain, M.: Side channel cryptanalysis of a higher order masking scheme. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 28–44. Springer, Heidelberg (2007)
13. Coron, J.-S., Prouff, E., Rivain, M., Roche, T.: Higher-order side channel security and mask refreshing. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 410–424. Springer, Heidelberg (2014)
14. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014)
15. Eldib, H., Wang, C.: Synthesis of masking countermeasures against side channel attacks. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 114–130. Springer, Heidelberg (2014)
16. Eldib, H., Wang, C., Schaumont, P.: SMT-based verification of software countermeasures against side-channel attacks. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 62–77. Springer, Heidelberg (2014)

17. Eldib, H., Wang, C., Taha, M.M.I., Schaumont, P.: QMS: evaluating the side-channel resistance of masked software from source code. In: The 51st Annual Design Automation Conference 2014, DAC 2014, San Francisco, CA, USA, June 1–5, pp. 1–6. ACM (2014)
18. Faust, S., Rabin, T., Reyzin, L., Tromer, E., Vaikuntanathan, V.: Protecting circuits from leakage: the computationally-bounded and noisy cases. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 135–156. Springer, Heidelberg (2010)
19. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
20. Gomes, C.P., Sabharwal, A., Selman, B.: Model counting. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 633–654. IOS Press (2009)
21. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
22. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
23. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
24. Micali, S., Reyzin, L.: Physically observable cryptography (extended abstract). In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004)
25. Moss, A., Oswald, E., Page, D., Tunstall, M.: Compiler assisted masking. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 58–75. Springer, Heidelberg (2012)
26. Oswald, E., Mangard, S.: Template attacks on masking—resistance is futile. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 243–256. Springer, Heidelberg (2007)
27. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A side-channel analysis resistant description of the AES S-box. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 413–423. Springer, Heidelberg (2005)
28. Peeters, E., Standaert, F.-X., Donckers, N., Quisquater, J.-J.: Improved higher-order side-channel attacks with FPGA experiments. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 309–323. Springer, Heidelberg (2005)
29. Pettai, M., Laud, P.: Automatic proofs of privacy of secure multi-party computation protocols against active adversaries. Cryptology ePrint Archive, Report 2014/240 (2014). <http://eprint.iacr.org/2014/240>
30. Prouff, E., Rivain, M.: Masking against side-channel attacks: a formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013)
31. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
32. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications **21**(1), 5–19 (2003)
33. Schramm, K., Paar, C.: Higher order masking of the AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 208–225. Springer, Heidelberg (2006)
34. Veyrat-Charvillon, N., Gérard, B., Standaert, F.-X.: Soft analytical side-channel attacks. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, PART I. LNCS, vol. 8873, pp. 282–296. Springer, Heidelberg (2014)

Inner Product Masking Revisited

Josep Balasch¹✉, Sebastian Faust^{2,3}, and Benedikt Gierlichs¹

¹ KU Leuven Department Electrical Engineering-ESAT/COSIC and iMinds,
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium

{Josep.Balasch,Benedikt.Gierlichs}@esat.kuleuven.be

² EPFL Lausanne, Lausanne, Switzerland

³ Ruhr-University Bochum, Bochum, Germany

sebastian.faust@gmail.com

Abstract. Masking is a popular countermeasure against side channel attacks. Many practical works use Boolean masking because of its simplicity, ease of implementation and comparably low performance overhead. Some recent works have explored masking schemes with higher algebraic complexity and have shown that they provide more security than Boolean masking at the cost of higher overheads. In particular, masking based on the inner product was shown to be practical, albeit not efficient, for a small security parameter, and at the same time provable secure in the domain of leakage resilient cryptography for a large security parameter. In this work we explore a security versus efficiency tradeoff and provide an improved and tweaked inner product masking. Our practical security evaluation shows that it is less secure than the original inner product masking but more secure than Boolean masking. Our performance evaluation shows that our scheme is only four times slower than Boolean masking and more than two times faster than the original inner product masking. Besides the practical security analysis we prove the security of our scheme and its masked operations in the threshold probing model.

1 Introduction

Side-channel attacks (SCA) are a well-known threat to embedded security. They allow to perform key recovery attacks on cryptographic implementations by analyzing physical properties present in embedded devices. Examples are execution time [22], power consumption [23] or electromagnetic emanations [16, 30]. SCA exploit the fact that these measurable quantities are statistically dependent on the intermediate variables being processed in the implementation. One of the most popular and well-studied countermeasures for block ciphers are data randomization techniques, commonly known as masking [5, 19]. These aim to conceal all intermediate variables of a cryptographic computation with random data.

The core principle of higher-order masking is to split any sensitive variable S into n random and secret shares. The way in which such a splitting is made determines the *masking type* or the *masking function*. Typical examples are Boolean masking ($S = S_1 + \dots + S_n$) or multiplicative masking ($S = S_1 \times \dots \times S_n$). A

masking scheme additionally defines a set of operations to process the n shares while preserving the correctness of computations and ensuring that intermediate values remain independent of the sensitive variables. The *security order* d of a masking scheme is defined as the smallest number of $d + 1$ intermediate values that, considered jointly, are *not* independent of a sensitive variable S . While a d -th order masking scheme can always be broken by a $d + 1$ -th order SCA that exploits the leakage of $d + 1$ intermediate values in the protected implementation jointly, the design of higher-order masking countermeasures is of practical interest due to two main reasons. First, the data complexity of applying a $d + 1$ -th order SCA grows exponentially on d given a sufficient amount of noise [5, 11, 27]. And second, the computational complexity of searching for the $d + 1$ leakage points grows combinatorially in the attack order [32].

1.1 Related Work

Ishai, Sahai and Wagner [21] were first to introduce a higher-order Boolean masking scheme tailored to hardware contexts by showing how to protect a circuit over \mathbb{F}_2 composed of NOT and AND gates. They also proved the security of their construction against an adversary capable of probing d wires of the protected circuit. The framework introduced in [21] (commonly known as the ISW probing model) provides a sound basis to determine the security of higher-order masking schemes, as security against d probes directly implies d -th order SCA resistance [7]. Ishai *et al.* proved their construction secure in the probing model for $n \geq 2d + 1$.

Rivain and Prouff [33] extended the ideas of [21] to any finite field. They devised a series of provable secure operations in the masked domain and applied them to efficiently secure AES implementations in software contexts. To protect the most complex part of AES, namely the nonlinear part of the S-box, they employed a power function devised to minimize the number of costly multiplications. Although the original claim in [33] was that $n = d + 1$ shares could provide d -th order provable security, Coron *et al.* [9] have shown that in fact $n \geq 2d + 1$ shares are necessary. Despite this, the Boolean scheme due to Rivain and Prouff remains one of the most efficient generic higher-order constructions in the literature.

Further work has focused on improving the performance of higher-order Boolean masking for the most challenging part of cipher implementations, namely nonlinear transformations. Genelle *et al.* [17] proposed a method to securely switch between Boolean and multiplicative masking at any order. The technique is particularly suitable to protect implementations of the AES, as it enables to switch the mask type before and after the S-box power function. Carlet *et al.* [4] built on the ideas of [33] to secure any look-up table using Lagrange interpolation. Coron [7] introduced a method to mask look-up tables at any order.

Generic higher-order countermeasures using other types of masking have also been investigated. Because of their higher algebraic complexity, observation of the shares results in significantly less information leakage than for the Boolean type given the same security order d and low levels of noise. Along these lines,

von Willich [37] proposed *affine* masking in which variables are encoded as $S' = \text{mask}_1 \times S + \text{mask}_2$. Fumaroli *et al.* [15] analyzed this type of masking and showed how it can be used to secure AES software implementations, in which the S-box is protected by means of table re-computation. The authors achieve performance results that are comparable to those of Boolean masking, but the affine masking construction has not been generalized to higher security orders $d > 1$.

Another proposal for a different type of masking is *polynomial* masking. Independently introduced by Prouff and Roche [29] as well as by Goubin and Martinelli [18], it employs Shamir's secret-sharing [34] and secure multi-party computation techniques [2]. In particular, a sensitive variable S is associated to a polynomial of degree d of the form $P_S(X) = S + \sum_{i=1}^d a_i \times X^i$ where the a_i are random secret coefficients. An encoding of a variable S is performed by selecting n distinct nonzero elements α_i and evaluating $S_i = P_S(\alpha_i)$ for $i = 1, \dots, n$. The variable S is then represented in the masked domain as the combination of n pairs (α_i, S_i) , of which only the S_i are secret. The variable S can be reconstructed as $S = \sum_{i=1}^n S_i \times \beta_i$, where the coefficients β_i are computed from the public α_i .

Although based on the same masking type, the schemes due to Prouff and Roche [29] and by Goubin and Martinelli [18] have notable differences. In particular, the construction of [29] is specifically designed to prove d -th order security in the presence of *glitches* [24], while the construction of [18] is designed to achieve "classical" d -th order SCA resistance. Both schemes use an algorithm to compute the product of two masked variables that is based on the secure multi-party computation scheme due to Ben-Or *et al.* [2], and that requires $n \geq 2d+1$. The authors of [18] additionally propose a more efficient algorithm that reduces the number of required shares to $n \geq d+1$. Despite this improvement, the complexity of both multiplication algorithms is $\mathcal{O}(n^3)$ in the number of shares, as opposed to e.g. Boolean masking which achieves $\mathcal{O}(n^2)$. A recent work by Coron *et al.* [8] has improved this complexity to $\tilde{\mathcal{O}}(n^2)$ by using DFT for fast polynomial evaluation.

A different approach is followed by Balasch *et al.* [1]. They introduce *IP* masking based on the inner product construction of Dziembowski and Faust [12]. A masked variable is represented by $2n$ shares in the form of two random vectors (\mathbf{L}, \mathbf{R}) of n elements each such that S equals the inner product of \mathbf{L} and \mathbf{R} . The authors propose an efficient multiplication algorithm with complexity $\mathcal{O}(n^2)$, thus similar to Boolean constructions, while achieving significantly less information leakage than other types of masking at the same security order d for low noise levels. Despite these advantages, the addition and refreshing algorithms are more complex (larger constant terms) than their counterparts in the Boolean and polynomial masking schemes.

Overall, the applicability of higher-order masking techniques other than Boolean is still an open question. Coron *et al.* [8] have identified a first-order flaw in the faster multiplication routine of the polynomial masking construction of Goubin and Martinelli [18]. Similarly, a first-order leakage in the refresh and addition operations of IP Masking [1] has been pointed out by Prouff *et al.* [28]. And the polynomial masking scheme by Prouff and Roche [29] has been shown

to be rather demanding when implemented in hardware [25], mostly due to its security guarantees even in the presence of glitches. Eventually, and despite their potential, higher-order countermeasures based on masking constructions other than Boolean do not appear to be ready for practical applications.

1.2 Our Contributions

In this work, we develop several improvements to the original IP masking scheme proposed in [1].

New IP Masking Scheme. Our first contribution is to introduce a few tweaks in the definition of the masking function that result in significant performance improvements. Similar to the original IP masking, a masked variable is represented by $2n$ shares in the form of two vectors (\mathbf{L}, \mathbf{R}) . Our first change is to let \mathbf{L} be a public value, allowing us to reduce the number of secret shares from $2n$ to n . As a result we fix \mathbf{L} to a constant value in such a way that all variables involved in computations are masked under the same \mathbf{L} , but different \mathbf{R} . Additionally, we require that the first public element of \mathbf{L} is $L_1 = 1$. The combination of these changes results in great efficiency improvements for all operations in the masked domain. In particular, the complexity of the addition and refreshing algorithms becomes comparable to those of Boolean and polynomial masking schemes. An important side benefit of our tweaks is that the first-order leakage identified in [28] on the refresh and addition algorithms no longer applies.

Practical Security Analysis. Our second contribution is to evaluate the impact of our tweaks and compare the security of the new masking type with other higher-order masking functions. We use the mutual information between the secret variable and the leakage of all shares of its masked representation as figure of merit. Our evaluation shows that our new masking function leaks more than IP masking, which is expected because \mathbf{L} is now public, but it leaks roughly one order of magnitude less than Boolean masking with the same security order d . It also leaks similar to polynomial masking with the same security order d .

Security in the Threshold-Probing Model. As a third contribution we prove the security of our improved scheme in the probing model introduced by Ishai, Sahai and Wagner [21]. Our security analysis shows that our construction is secure against d probes, when $n \geq 2d + 1$. We emphasize that this is the same security threshold that can be achieved by most other higher-order masking schemes [7, 21, 33]. Notice that only for the multiplication operation we require that $n \geq 2d + 1$. For all other operations including the masking function, it is sufficient to set $n > d$.

Efficient Implementation. Finally, our fourth contribution is to determine the performance of our masking scheme in securing a block cipher implementation. Similar to [1], we opt to protect a software implementation of AES-128 on an

embedded 8-bit controller and we compare our results to other d -th order masking schemes. The results show that our improved construction allows to halve the execution time with respect to the original IP masking scheme, and to reduce the gap with Boolean masking from approximately a factor 10 to a factor 4.

2 Notation

In the following we denote by \mathcal{K} a field of characteristic 2 and we represent field elements with upper-case letters. For instance, $S \in \mathbb{F}_{2^8}$ denotes an element in the AES field $GF(2^8)$. Let \mathbf{X}, \mathbf{Y} represent two vectors over \mathcal{K}^n . Field elements in vectors are addressed with subindex i , e.g. X_i and Y_i , respectively. The standard inner product function over \mathcal{K} is denoted as $\langle \mathbf{X}, \mathbf{Y} \rangle = \sum_i X_i \times Y_i$.

The matrix $\hat{\mathbf{A}} = \mathbf{X} \times \mathbf{Y}$ over $\mathcal{K}^{n \times n}$ is defined as the tensor product of two vectors \mathbf{X} and \mathbf{Y} . Field elements in a matrix are addressed as $A_{i,j}$, where i and j represent row and column position, respectively. If $\hat{\mathbf{A}}, \hat{\mathbf{B}}$ are matrices over $\mathcal{K}^{n \times n}$, then we denote by $\langle \hat{\mathbf{A}}, \hat{\mathbf{B}} \rangle$ the inner product of matrices when we view them as vectors of n^2 field elements, i.e. $\sum_i \sum_j A_{i,j} \times B_{i,j}$.

For an integer n we denote by $[n]$ the set $\{1, \dots, n\}$.

3 Our Construction

Our scheme improves on the IP masking by Balasch *et al.* [1] based on the inner product construction of Dziembowski and Faust [12]. A variable $S \in \mathcal{K}$ in IP masking is encoded by using two vectors (\mathbf{L}, \mathbf{R}) of n elements with $\mathbf{L} \leftarrow \mathcal{K} \setminus \{0\}^n$ and $\mathbf{R} \leftarrow \mathcal{K}^n$. Note that the elements of the vector \mathbf{L} are by definition different than zero.

Our new masking function has three major differences with respect to [1]. First, the vector \mathbf{L} is computed once and kept as a constant parameter. This implies that all masked variables employed in our scheme share a unique vector \mathbf{L} . Second, we let the vector \mathbf{L} be a public (rather than secret) parameter. In other words, we assume the elements L_i can be known to the adversary. And third, we constrain the selection of the first element of \mathbf{L} such that $L_1 = 1$. The number of shares that are kept secret in our masking function is therefore determined by the security parameter n , which corresponds to the number of elements in \mathbf{R} . We show in the remainder of this section that all these choices allow to significantly reduce the complexity of operations in the masked domain.

The procedure IPSetup_n depicted in Algorithm 1 details the initialization steps of our masking construction. Given n and a field description \mathcal{K} , the algorithm returns a public vector \mathbf{L} and a public matrix $\hat{\mathbf{L}}$. The latter corresponds to the tensor product $\mathbf{L} \times \mathbf{L}$, and its pre-computation allows us to speed-up multiplications in the masked domain. One can imagine IPSetup_n is executed before system roll-out during device personalization, e.g. in parallel with key generation. The sub-routine `randNonZero()` returns a nonzero element in the field \mathcal{K} . More precisely, it samples uniformly at random from $\mathcal{K} \setminus \{0\}$.

Algorithm 1. Setup the masking scheme: $(\mathbf{L}, \hat{\mathbf{L}}) \leftarrow \text{IPSetup}_n(\mathcal{K})$

Input: field description \mathcal{K} **Output:** random vector \mathbf{L} and tensor product $\hat{\mathbf{L}} = \mathbf{L} \times \mathbf{L}$

```

1:  $L_1 = 1;$ 
2: for  $i = 2$  to  $n$  do
3:    $L_i \leftarrow \text{randNonZero}(\mathcal{K});$ 
4: end for
5: for  $i = 1$  to  $n$  do
6:   for  $j = 1$  to  $n$  do
7:      $L_{i,j} = L_i \times L_j;$ 
8:   end for
9: end for

```

Algorithm 2 depicts the steps to convert a variable $S \in \mathcal{K}$ into the masked domain. This routine, denoted by $\text{IPMask}_{\mathbf{L}}$, is parametrized by a vector \mathbf{L} resulting from executing IPSetup_n . The sub-routine $\text{rand}()$ returns a randomly selected element in the field \mathcal{K} . This function is called $n - 1$ times in order to set the values of $R_2 \dots R_n$. The value R_1 is then computed in order to obtain a valid masking of S under the inner product construction.

Algorithm 2. Masking a variable: $\mathbf{R} \leftarrow \text{IPMask}_{\mathbf{L}}(S)$

Input: variable $S \in \mathcal{K}$ **Output:** vector \mathbf{R} such that $S = \langle \mathbf{L}, \mathbf{R} \rangle$

```

1: for  $i = 2$  to  $n$  do
2:    $R_i \leftarrow \text{rand}(\mathcal{K});$ 
3: end for
4:  $R_1 = S + \sum_{i=2}^n L_i \times R_i$ 

```

3.1 Operations in the Masked Domain

We propose three main algorithms of our masking construction: $\text{IPRefresh}_{\mathbf{L}}$, $\text{IPAdd}_{\mathbf{L}}$ and $\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}$. For an implementation of the AES (as detailed in Sect. 6) it is advantageous to further have a dedicated squaring routine in the masked domain. For this reason we also propose an additional $\text{IPSquare}_{\mathbf{L}}$ algorithm.

The routine $\text{IPRefresh}_{\mathbf{L}}$ is depicted in Algorithm 3. It consists of two steps. First, the computation of a vector \mathbf{A} orthogonal to \mathbf{L} . And second, the addition of \mathbf{A} to \mathbf{R} to obtain a fresh vector \mathbf{R}' . The correctness of the algorithm is easy to prove:

$$\langle \mathbf{L}, \mathbf{R}' \rangle = \langle \mathbf{L}, \mathbf{R} + \mathbf{A} \rangle = \langle \mathbf{L}, \mathbf{R} \rangle + \langle \mathbf{L}, \mathbf{A} \rangle = \langle \mathbf{L}, \mathbf{R} \rangle + 0 = \langle \mathbf{L}, \mathbf{R} \rangle.$$

The routine $\text{IPAdd}_{\mathbf{L}}$ is illustrated in Algorithm 4. Because all variables in our construction are masked with a common vector \mathbf{L} , the output vector \mathbf{T} can be simply obtained by adding the input vectors \mathbf{R} and \mathbf{Q} .

Algorithm 3. Refresh vector: $\mathbf{R}' \leftarrow \text{IPRefresh}_{\mathbf{L}}(\mathbf{R})$

Input: vector \mathbf{R} **Output:** vector \mathbf{R}' such that $\langle \mathbf{L}, \mathbf{R} \rangle = \langle \mathbf{L}, \mathbf{R}' \rangle$ 1: $\mathbf{A} \in_R \mathcal{K}^n$ s.t. $\langle \mathbf{A}, \mathbf{L} \rangle = 0$ 2: $\mathbf{R}' = \mathbf{R} + \mathbf{A}$

Algorithm 4. Add masked values: $\mathbf{T} \leftarrow \text{IPAdd}_{\mathbf{L}}(\mathbf{R}, \mathbf{Q})$

Input: vectors \mathbf{R} and \mathbf{Q} **Output:** vector \mathbf{T} such that $\langle \mathbf{L}, \mathbf{T} \rangle = \langle \mathbf{L}, \mathbf{R} \rangle + \langle \mathbf{L}, \mathbf{Q} \rangle$ 1: $\mathbf{T} = \mathbf{R} + \mathbf{Q}$

Note that the $\text{IPAdd}_{\mathbf{L}}$ algorithm is similar to that of Boolean masking constructions, yet our type of masking has higher algebraic complexity. This improvement is a direct consequence of letting \mathbf{L} be a public and constant parameter.

The multiplication routine $\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}$ depicted in Algorithm 5 is the most involved operation. Our starting point is the masked multiplication of [1], albeit with some efficiency improvements. First, since both input operands are masked under the same vector \mathbf{L} , the computation of the matrix $\hat{\mathbf{L}}$ is not dependent on the input operands. Consequently, we can save n^2 field multiplications by pre-computing this matrix during IPSetup_n . And second, because the first component of \mathbf{L} is set to $L_1 = 1$, a constant b can be added to a masking (\mathbf{L}, \mathbf{R}) by simply computing the new masking as $(\mathbf{L}, (R_1 + b, R_2, \dots, R_n))$.

Algorithm 5. Multiply masked values: $\mathbf{T} \leftarrow \text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}(\mathbf{R}, \mathbf{Q})$

Input: vectors \mathbf{R} and \mathbf{Q} **Output:** vector \mathbf{T} such that $\langle \mathbf{L}, \mathbf{T} \rangle = \langle \mathbf{L}, \mathbf{R} \rangle \times \langle \mathbf{L}, \mathbf{Q} \rangle$ 1: $\hat{\mathbf{A}} \in_R \mathcal{K}^{n \times n}$ s.t. $\langle \hat{\mathbf{L}}, \hat{\mathbf{A}} \rangle = 0$ 2: $\hat{\mathbf{R}} = \mathbf{R} \times \mathbf{Q}$ 3: $\hat{\mathbf{B}} = \hat{\mathbf{R}} \oplus \hat{\mathbf{A}}$ 4: $b = \sum_{i=2}^n \sum_{j=1}^n L_{i,j} \times B_{i,j}$ 5: $\mathbf{T} = (B_{1,1} + b, B_{1,2}, \dots, B_{1,n})$

In order to keep the d -th order security for $n = 2d + 1$ throughout the whole execution of $\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}$, it is important that operations in lines 1 and 4 are computed in a certain way as depicted in Table 1. In particular, the intermediate values Δ_j are calculated by aggregating the intermediate products of elements in matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{L}}$ in a column-wise fashion. In contrast, the values β_i are computed by processing the elements of the matrices $\hat{\mathbf{L}}$ and $\hat{\mathbf{B}}$ row by row. This important difference in the way we compute the sums is crucial for the security proof and, in fact, crucial for the actual security of our scheme.

For illustration purposes, consider a setting where the operations in line 1 and line 4 of $\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}$ are computed row-wise. Let us also assume that all elements in \mathbf{L} are $L_i = 1$. Under these circumstances, the following attack would apply:

Table 1. Detailed description of operations in $\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}$

LINE 1: $\hat{\mathbf{A}} \in_R \mathcal{K}^{n \times n}$ s.t. $\langle \hat{\mathbf{L}}, \hat{\mathbf{A}} \rangle = 0$	LINE 4: $b = \sum_{i=2}^n \sum_{j=1}^n L_{i,j} \times B_{i,j}$
$\text{// Random sampling}$ for $(i, j) \neq (n, n)$ do $A_{i,j} \leftarrow \text{rand}(\mathcal{K})$ end $\text{// Column-wise processing}$ $\Delta_0 = 0$ for $j = 1$ to $n - 1$ do $\Delta_j = \Delta_{j-1} + \sum_{i=1}^n A_{i,j} \times L_{i,j}$ end $A_{n,n} = (\Delta_{n-1} + \sum_{i=1}^{n-1} A_{i,n} \times L_{i,n}) \times L_{n,n}^{-1}$	$\text{// Row-wise processing}$ $\beta_1 = 0$ for $i = 2$ to n do $\beta_i = \beta_{i-1} + \sum_{j=1}^n L_{i,j} \times B_{i,j}$ end $b = \beta_n$

1. The adversary learns the value $\Delta_2 = \sum_{j=1}^n A_{2,j}$
2. The adversary learns $\beta_2 = \sum_{j=1}^n A_{2,j} + Q_2 \times R_j = \Delta_2 + Q_2 \langle \mathbf{L}, \mathbf{R} \rangle$
3. The adversary learns Q_2

Assuming that $Q_2 \neq 0$ the above attack indeed recovers completely the secret value $\langle \mathbf{L}, \mathbf{R} \rangle$. Notice that the attack even applies when $L_i \neq 1$ but in this case the bias in the leaky distribution decreases with the number of shares. We prevent this attack by computing the intermediate values Δ_j as a sum of elements in a column, and β_i as a sum of elements in a row. This approach effectively results in a “mixing” of the random shares and enables a security proof.

For an implementation of the AES it is beneficial to have a particularly efficient implementation of the squaring algorithm. The $\text{IPSquare}_{\mathbf{L}}$ routine is illustrated in Algorithm 6.

Algorithm 6. Square masked variable: $\mathbf{T} \leftarrow \text{IPSquare}_{\mathbf{L}}(\mathbf{R})$

Input: vector \mathbf{R}

Output: vector \mathbf{R}' such that $\langle \mathbf{L}, \mathbf{T} \rangle = \langle \mathbf{L}, \mathbf{R} \rangle \times \langle \mathbf{L}, \mathbf{R} \rangle$

```

for  $i = 1$  to  $n$  do
     $T_i \leftarrow (R_i)^2 \times L_i;$ 
end for

```

4 Practical Evaluation

In this section we evaluate the information leakage of our improved IP masking scheme and compare it to that of other masking schemes that can be implemented at any order, e.g. Boolean masking, polynomial masking and the original IP masking. We follow the common approach to focus the analysis on the type of masking, i.e. to analyze the leakage of the shares of one masked variable. In practice, the leakage of the operations in the masked domain depends a lot on

their (secure) implementation and on the target platform, for instance a table lookup in software versus combinational logic in hardware. We abstract from such practical issues to be able to provide a fair and meaningful comparison.

4.1 Attack Order

We begin the evaluation by deriving the minimum order for an attack against our type of masking. We say that a masking function is d -th order SCA secure, if every tuple of d or less shares is independent of the masked variable.

It is easy to see that the masking function of our improved scheme is d -th order SCA secure for $n = d + 1$. If we choose all elements in \mathbf{L} equal to 1 the argument is exactly the same as for Boolean masking. It uses the same number of uniformly distributed secret shares. If we choose any L_i greater than 1 we just need to observe that the field multiplication $L_i \times R_i$ does not introduce biases, i.e. for uniformly distributed R_i the output is uniformly distributed. Recall that all elements in \mathbf{L} are nonzero by definition.

4.2 Information Leakage

It remains to explore the security versus efficiency tradeoff of our improved scheme. It is known that a more complex algebraic relation between the shares and the masked variable provides less information leakage. We hence expect our type of masking to leak less information than the Boolean type whenever at least one L_i is unequal to 1 (since the field multiplication $L_i \times R_i$ adds diffusion) but more than the original IP masking since \mathbf{L} is public.

Similar to our type of masking, in polynomial masking half of the shares are distinct nonzero public constants and the other half are random and secret masks. For our evaluation of information leakage we refer only to the n secret shares. For example, polynomial masking of security order d uses $n = d + 1$ random and secret shares, and can theoretically be broken by a $d + 1$ -th order SCA. Due to the similar representations of variables in the masked domain, we expect our masking function and the polynomial type to provide comparable information leakage.

We compare our type of masking with Boolean masking, polynomial masking (all of security order d using $n = d + 1$ secret shares) and, for completeness, with the original IP masking (of security order $d = 1$ using $n = 4$ secret shares).

Following previous work [1, 18, 29, 35, 36] we use the mutual information between a variable and the leakage of all shares of its masked representation as criterion for the comparison. We estimate the mutual information using computer simulations.

We evaluate our improved IP masking for $d = 1$ ($L_1 = 1$, $L_2 = 255$) and for $d = 2$ ($L_1 = 1$, $L_2 = 15$, $L_3 = 233$). Boolean masking uses $d + 1$ shares (M_1, \dots, M_d, V) where the $M_i \in_R \mathbb{F}_{2^8}$ and V is computed such that $S = M_1 + \dots + M_d + V$ holds. We evaluate Boolean masking for $d \in \{1, 2, 3\}$. Polynomial masking uses $d + 1$ public coefficients $(\alpha_1, \dots, \alpha_{d+1})$ with $\alpha_i \in_R \mathbb{F}_{2^8} \setminus \{0\}$ and pairwise distinct, and $d + 1$ shares (S_1, \dots, S_{d+1}) with $S_i = P_S(\alpha_i) \in \mathbb{F}_{2^8}$ [29].

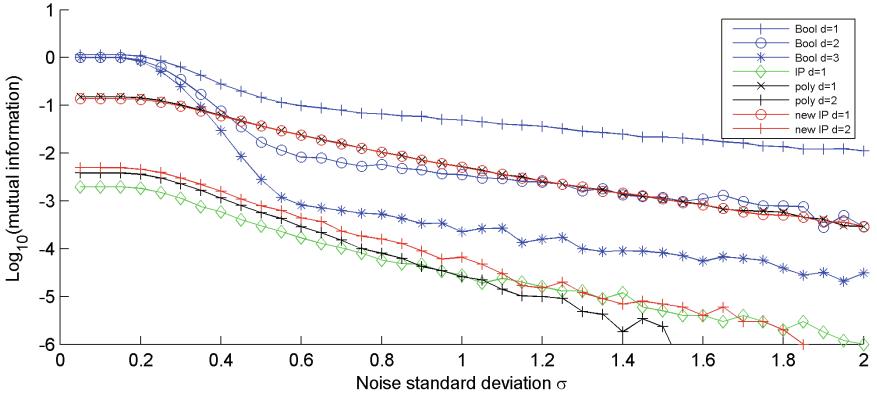


Fig. 1. Mutual information (\log_{10}) over increasing noise standard deviation σ for different masking functions

We evaluate polynomial masking for $d = 1$ ($\alpha_1 = 3, \alpha_2 = 7$) and $d = 2$ ($\alpha_1 = 13, \alpha_2 = 240, \alpha_3 = 163$). For the original IP masking we set $d = 1$ and let $R_2 \in_R \mathbb{F}_{2^8}$ and $L_1, L_2 \in_R \mathbb{F}_{2^8} \setminus \{0\}$ such that $S = L_1 \times R_1 + L_2 \times R_2$.

We model the relation between a share and its physical leakage as usual in the literature: each share leaks its Hamming weight, each share leaks independently of all other shares, and each leakage is affected by independent Gaussian noise. In summary, we model the leakage of our improved scheme as

$$\text{Leak}(\mathbf{L}, \mathbf{R}) = (\text{HW}(R_1) + n_1, \dots, \text{HW}(R_{d+1}) + n_{d+1}),$$

the leakage of boolean masking as

$$\text{Leak}(M_1, \dots, M_d, V) = (\text{HW}(M_1) + n_1, \dots, \text{HW}(M_d) + n_d, \text{HW}(V) + n_{d+1}),$$

the leakage of polynomial masking as

$$\text{Leak}(S_1, \dots, S_{d+1}) = (\text{HW}(S_1) + n_1, \dots, \text{HW}(S_{d+1}) + n_{d+1})$$

and the leakage of the original IP masking as

$$\text{Leak}(\mathbf{L}, \mathbf{R}) = (\text{HW}(L_1) + n_1, \text{HW}(R_1) + n_2, \text{HW}(L_2) + n_3, \text{HW}(R_2) + n_4),$$

where the n_i are independent Gaussian variables with mean zero and standard deviation σ . The mutual information between the secret variable and the leaked information is then $I(S; \text{Leak}(\mathbf{L}, \mathbf{R}))$, $I(S; \text{Leak}(M_1, \dots, M_d, V))$, and $I(S; \text{Leak}(S_1, \dots, S_{d+1}))$ respectively. Recall that the mutual information is directly related to the number of measurements that a Template Attack [6] (worst case attack scenario) requires to achieve a given success probability. Standaert et al. [36] defined the relation via $c \cdot I(\cdot; \cdot)^{-1}$ where the constant c is related to the success probability.

Figure 1 shows plots of the mutual information (in \log_{10} scale) between S and the information leaked by all shares of its masked representation, over increasing noise levels σ , for all masking types considered.

The results are in line with our expectations. Our improved IP masking leaks more information than the original IP masking (in the improved scheme \mathbf{L} is public), which illustrates the security versus efficiency tradeoff. But it leaks less than Boolean masking with the same number of shares, which is due to the more complex algebraic relation between the shares and the secret variable. The difference is particularly pronounced for low levels of noise and it seems to increase with increasing security order d . Finally, our improved IP masking and polynomial masking leak comparably. We attribute this to their similar representation of variables in the masked domain.

5 Security Proof in the Probing Model

We start our security analysis with a proof in the so-called probing model introduced by Ishai, Sahai and Wagner [21]. Recall that our masking scheme has the form (\mathbf{L}, \mathbf{R}) , where \mathbf{L} is public and \mathbf{R} is secret and random in \mathcal{K}^n for $n \in \mathbb{N}$ being the security parameter. We will show that our construction is secure against any d probing adversary, where we assume that $n = 2d+1$. That is, an adversary that can learn up to d arbitrary intermediate values computed during the execution of the masked scheme will not learn anything about the underlying secrets. Let in the following denote by \mathcal{P} the set of intermediate values that the adversary is probing.

As a first step we show that our masking function is indeed secure against an $(n - 1)$ -probing adversary. We use the notation $\mathcal{A}_{n-1}(\text{IPMask}_{\mathbf{L}}(S))$ to describe that the adversary \mathcal{A} obtains at most $n - 1$ shares of the masking.

Lemma 1. *For any two secrets $S, S' \in \mathcal{K}$ and any $(n - 1)$ -probing adversary \mathcal{A} we have*

$$\mathcal{A}_{n-1}(S) = \mathcal{A}_{n-1}(S'),$$

where \mathbf{L} was sampled as specified by the setup algorithm and $\mathbf{S} \leftarrow \text{IPMask}_{\mathbf{L}}(S)$ and $\mathbf{S}' \leftarrow \text{IPMask}_{\mathbf{L}}(S')$.

Proof. Notice that in our scheme the vector \mathbf{L} is public and hence does not contribute to the security against probing attacks. Further, recall that all $L_i \neq 0$, and hence all values of \mathbf{R} contribute to the security of the masking. The proof follows by the fact that given \mathbf{L} with all components $\neq 0$ the vector \mathbf{R} is a perfect random additive $(n - 1)$ out of n secret sharing scheme. More precisely, let $\mathcal{I} \subset [n]$ be the subset of indices for which \mathcal{A}_{n-1} learns R_i , i.e., for all $i \in \mathcal{I}$ we have that \mathcal{A}_{n-1} probes R_i . As $|\mathcal{I}| < n$ there exists at least one $j \in [n]$ such that $j \notin \mathcal{I}$. Hence, for any value $S \in \mathcal{K}$, any choice of \mathbf{L} and any R_i such that $i \in \mathcal{I}$ there exists R_j such that $S = \langle \mathbf{L}, \mathbf{R} \rangle$. \square

5.1 Security of Masked Operations

We now show the security of the different operations presented in Section 3. Informally, we will show that any subset of wires \mathcal{P} with size $|\mathcal{P}| \leq d$ is independent of the underlying masked values, i.e., the probes \mathcal{P} given to the adversary

will not help the adversary in breaking the security guarantee of the underlying scheme. To this end, we will prove the security for any masked operation individually and then show that also a combination of such masked operations remains secure if the adversary obtains at most d probes in the entire circuit.

The proof is easy for the masked addition operation: probes at the inputs and outputs directly translate to probes at the underlying masked value. We will show security for the masked squaring and masked multiplication operation. The proof of the masked multiplication is rather tedious since simulating the intermediate values just from the encoded inputs and outputs of the masked operations requires careful bookkeeping.

We will denote by $\mathcal{A}_d(\text{Operation}(\mathbf{X}, \mathbf{Y}))$ the output of an adversary that probes up to d values in the execution of the operation **Operation** when run on masked inputs \mathbf{X} and \mathbf{Y} . Notice that \mathcal{A}_d may also probe the output produced by **Operation**, i.e., $\mathbf{Z} \leftarrow \text{Operation}(\mathbf{X}, \mathbf{Y})$. As an example let **Operation** be the $\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}(\mathbf{Q}, \mathbf{R})$ operation on inputs \mathbf{Q} and \mathbf{R} . The adversary may learn up to d of the intermediate values produced during the computation of this algorithm. Moreover, for a vector \mathbf{X} and a subset $\mathcal{I} \subset [n]$ we denote by $\mathbf{X}|_{\mathcal{I}}$ the set $\{X_i\}_{i \in \mathcal{I}}$. Following Ishai et al. we will say that a masked operation **Operation** is secure against d probing attacks if probes on intermediate values produced by the masked operation **Operation** can be simulated by just access to the inputs of the operation.

Security of the masked squaring operation. It is simple to show security of the squaring algorithm presented in Algorithm 6 in the probing model, where $d < n$.

Lemma 2. *Let n be the security parameter and let $d < n$, then for any d -probing adversary \mathcal{A}_d and any $R \in \mathcal{K}$ and $\mathbf{R} \leftarrow \text{IPMask}_{\mathbf{L}}(R)$, there exists a subset $\mathcal{I} \subset [n]$ with $|\mathcal{I}| \leq d$ and a simulator $\text{Sim}(\mathbf{R}|_{\mathcal{I}})$ such that:*

$$\mathcal{A}_d(\text{IPSquare}_{\mathbf{L}}(\mathbf{R})) \equiv \text{Sim}(\mathbf{R}|_{\mathcal{I}}).$$

Proof. We start by a description of how to build the set \mathcal{I} , which initially is set to $\mathcal{I} = \{\}$. For each probe of the form $R_i, (R_i)^2$ or $(R_i)^2 \times L_i$ add the index i to \mathcal{I} . Since the adversary can make at most d probes, we clearly have $|\mathcal{I}| \leq d$. Given the set $\mathbf{R}_{\mathcal{I}}$ and \mathbf{L} (which is public and hence the simulator has access to it), it is easy to simulate all probes in $\text{IPSquare}_{\mathbf{L}}(\mathbf{R})$ in a perfect way and in particular consistent with probes on the real execution of the squaring algorithm. \square

Security of masked multiplication. We prove the security of Algorithm 5 in the d -probing model, where $n \geq 2d + 1$.

Lemma 3. *For any $Q, R \in \mathcal{K}$ let $\mathbf{Q} \leftarrow \text{IPMask}_{\mathbf{L}}(Q)$ and $\mathbf{R} \leftarrow \text{IPMask}_{\mathbf{L}}(R)$. Let n be the security parameter and let d be such that $2d < n$, then for any d -probing adversary \mathcal{A}_d that learns at most d probes on intermediate values produced during the masked multiplication $\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}(\mathbf{Q}, \mathbf{R})$, there exists a subset $\mathcal{I} \subset [n]$ with $|\mathcal{I}| \leq 2d$ and a simulator $\text{Sim}(\mathbf{Q}|_{\mathcal{I}}, \mathbf{R}|_{\mathcal{I}})$ such that:*

$$\mathcal{A}_d(\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}(\mathbf{Q}, \mathbf{R})) \equiv \text{Sim}(\mathbf{Q}|_{\mathcal{I}}, \mathbf{R}|_{\mathcal{I}}).$$

Proof. To simplify the analysis we assume that \mathbf{L} is given in its entirety a-priori to the adversary. We show that the entire distribution of the multiplication algorithm can be simulated by having access to at most $2d$ shares of \mathbf{Q} and \mathbf{R} , respectively. Since by Lemma 1 seeing $2d$ shares of \mathbf{Q} and \mathbf{R} respectively, is independent of the masked secret this proves the security of the masked multiplication operation. The set that keeps track of what values are revealed of \mathbf{Q}, \mathbf{R} is called \mathcal{I} . Moreover, we keep track of two sets \mathcal{T} and \mathcal{U} that are needed to keep the simulation consistent. \mathcal{T} keeps tuples $(i, j) \subset [n] \times [n]$ of pairs that correspond to the values of $A_{i,j}$ which are revealed during a probing attack, while \mathcal{U} represents the values (i, j) of $B_{i,j}$ that are revealed by the attack. To simplify our analysis, we will be rather generous to the adversary and usually give him much more values than what are revealed during the actual probe of the particular intermediate value. Below we describe how to build the set \mathcal{I} .

A. Building the sets $\mathcal{I}, \mathcal{U}, \mathcal{T}$: In this step we initialize the sets that later are needed for the simulation.

1. Initially, we set the sets $\mathcal{I}, \mathcal{U}, \mathcal{T}$ to the empty set $\{\}$.
2. *Probes when Δ_i is computed, i.e., probes of the form $A_{1,i}, \dots, A_{n,i}$ or probes of the form $\Delta_{i-1} + \sum_{1 \leq j < n} L_{j,i} \times A_{j,i}$:* Add the index $(1, i), \dots, (n, i)$ into the set \mathcal{T} .
3. *Probes of the form $B_{i,j}$ or sums of the form $\sum_j L_{i,j} \times B_{i,j}$:* We distinguish two cases:
 - (a) For $i > 1$: Add the index $(i, 1), \dots, (i, n)$ to the set \mathcal{U} .
 - (b) For $i = 1$: Add the index (i, j) to the set \mathcal{U} .
 The above two cases capture the fact that for row $i = 1$ of the matrix $\hat{\mathbf{B}}$ we do not compute the sum of values $B_{1,j}$, i.e., of the first row of the matrix $\hat{\mathbf{B}}$. Additionally, for each such (i, j) that has been added to \mathcal{U} : if (i, j) is in \mathcal{T} , then add i, j to \mathcal{I} .
4. *Probes of the form Q_i, R_j and $Q_i \times R_j$:* For probes of the form Q_i resp. R_j add i resp. j to \mathcal{T} . For a probe of the form $Q_i \times R_j$ add the indices i, j to \mathcal{I} .

Given the above description of the the sets \mathcal{I}, \mathcal{T} and \mathcal{U} , we can now define the simulator $\text{Sim}_{\mathcal{I}, \mathcal{T}, \mathcal{U}}(\mathbf{R}|_{\mathcal{I}}, \mathbf{Q}|_{\mathcal{I}})$.

B. Sampling variables independently of probes: We start by sampling some of the values a-priori before we answer the actual probing queries. We will later take care that all probes are answered consistently with the values sampled in this initial step.

1. Choose β_2, \dots, β_n uniformly at random. Notice that this allows to compute all the potential values that appear in the sum when computing the value b – including the value b .
2. Sample $\Delta_1, \dots, \Delta_{n-1}$ uniformly at random and set $\Delta_0 = \Delta_n = 0$.

C. Simulating the probes: We next show how to answer the probing queries of the adversary given all values $\{Q_i\}_{i \in \mathcal{I}}$ and $\{R_i\}_{i \in \mathcal{I}}$, and the values Δ_i, β_j sampled in Step B.

1. *Probes of the form β_2, \dots, β_n and sums thereof:* These values have been fixed in Step B1 and hence probes on these values can easily be answered from the above sampled values.
2. *Probes on the sampling of $A_{i,j}$:* Notice that this involves the individual values $A_{i,j}$ as well as sub-sums of values in the columns with the appropriate values Δ_i . If $(1, i), \dots, (n, i)$ are in \mathcal{T} then sample $A_{1,i}, \dots, A_{n,i}$ uniformly at random such that $\sum_j L_{j,i} \times A_{j,i} = \Delta_i + \Delta_{i-1}$. Given the above sampled values and the values Δ_i sampled in Step B2, we can answer any probe of the adversary.
3. *Probes of the form Q_i, R_j and $Q_i \times R_j$:* Given access to $\{Q_i\}_{i \in \mathcal{I}}$ and $\{R_i\}_{i \in \mathcal{I}}$ we can easily simulate the probes in a consistent way.
4. *Probes of $B_{i,j}$, or when $i > 1$ of sums thereof:* To answer these probes, we sample the values of $B_{i,j}$ in the following way:
 - (a) If (i, j) is not in \mathcal{U} then leave the values $B_{i,j}$ un-assigned.
 - (b) If (i, j) is in \mathcal{U} and (i, j) is in \mathcal{T} then compute $B_{i,j} = Q_i \times R_j + A_{i,j}$. Notice that this is possible since the relevant values of Q_i and R_j are given in $\{Q_i\}_{i \in \mathcal{I}}$ and $\{R_i\}_{i \in \mathcal{I}}$ and $A_{i,j}$ has been assigned in Step C2.
 - (c) If (i, j) is in \mathcal{U} , but (i, j) is not in \mathcal{T} , then we sample $B_{i,j}$ uniformly at random subject to the constraint that $\beta_i = \sum_j L_{i,j} \times B_{i,j}$. Notice that the later requirement only is needed for $i > 1$. The value $B_{1,j}$ is chosen uniformly at random.

We will show below that (1) the simulation has the same distribution as the real execution of the masked multiplication operation (second claim below), and (2) we argue that the size of the set \mathcal{I} has always cardinality $|\mathcal{I}| \leq 2d$ (first claim below). Putting these two claims together proves the lemma.

In the following analysis we denote by u the number of probes corresponding to Step A2, by v the number of probes corresponding to Step A3 and by w the number of probes corresponding to Step A4.

Claim. Let $n \in \mathbb{N}$ be the security parameters and d be the number of probes such that $2d < n$. Then, $|\mathcal{I}| \leq 2d$.

Proof. Observe that the simulator adds elements to \mathcal{I} only in Step A3 and Step A4. As each probe in Step A4 leads to adding at most two elements to \mathcal{I} and $w \leq d$, this directly implies $|\mathcal{I}| \leq 2d$ if probes appear only in Step A4. It remains to analyze the number of elements we add to \mathcal{I} for each probe done in Step A3. The analysis for Step A3 is a little more involved as the number of elements added to \mathcal{I} depends on both u (number of probes in Step A2) and v (number of probes in Step A3). Recall that for each probe of Step A2 that is within the i -th column we add all indices $(1, i), \dots, (n, i)$ to \mathcal{T} that correspond to the elements in the i -th column of the matrix $\hat{\mathbf{A}}$, while for each probe in Step A3 we add all indices $(i, 1), \dots, (i, n)$ to \mathcal{U} that correspond to the i -th row of the matrix $\hat{\mathbf{B}}$ (except for the first row, but this does not matter for the rest of the analysis). Furthermore, recall that each $B_{i,j}$ is computed from $A_{i,j}$. Depending on the values added to \mathcal{T} and \mathcal{U} , in Step A3 we will add all i, j to \mathcal{I} where (i, j) is an “intersection” between the columns and rows mentioned above.

Unfortunately, it is easy to see that for u columns and $v = d - u - w$ rows we may have more than $2d$ intersections.¹ The good news is, however, that many elements will be added multiple times to \mathcal{I} , hence not increasing the size of \mathcal{I} .

More precisely, for a query in the i -th row from Step A3, we add the index i into the set \mathcal{I} . Additionally, for each such query we add index of the column at which we have an intersection from a query in the j -th column. The main observation is that for each row the indices added by the intersections with the columns are the same. In other words, for each i the tuples (i, j) have the same second component, and hence we add at most $u + v \leq d$ indices to \mathcal{I} in Step A3. Combining it with the probes from Step A4, we add $u + v + 2w$ elements to \mathcal{I} . Since $u + v + w \leq d$, we get that $|\mathcal{I}| \leq 2d$, which proves the claim. \square

The next claim shows that the sampling of the simulator produces the same view as a d -probing adversary obtains in a real attack against the masked multiplication operation.

Claim. For any $Q, R \in \mathcal{K}$ let $\mathbf{Q} \leftarrow \text{IPMask}_L(Q)$ and $\mathbf{R} \leftarrow \text{IPMask}_L(R)$, we have:

$$\mathcal{A}_d(\text{IPMult}_{\mathbf{L}, \hat{\mathbf{L}}}(\mathbf{Q}, \mathbf{R})) \equiv \text{Sim}(\mathbf{Q}|_{\mathcal{I}}, \mathbf{R}|_{\mathcal{I}}),$$

with parameters defined as in the statement of the lemma.

Proof. By the last claim we have $|\mathcal{I}| \leq 2d$. We compare the way in which the probes are answered by the simulator in Steps B and C with the real attack against the execution of the masked multiplication.

Step B: The joint distribution of values sampled at this step in the simulation is identically distributed with the real experiment even given all these values to the adversary. This is easy to see for the β_i values as they are just sums of random values $A_{i,j}$. Moreover, for the Δ_j values we observe that the first row is never used in computing β_i (since $i > 1$), and hence all Δ_j for $j < n$ can be chosen uniformly at random (they can essentially be made consistent with the view of the adversary by choosing $A_{1,j}$ appropriately. The value Δ_n is fixed to 0 as we require $\langle \hat{\mathbf{A}}, \hat{\mathbf{L}} \rangle = 0$.

Step C2: In Step B2 we sampled $\Delta_1, \dots, \Delta_{n-1}$ uniformly at random and in Step C2 we sample each column $(A_{1,i}, \dots, A_{n,i})$ of the matrix $\hat{\mathbf{A}}$ uniformly at random such that $\Delta_i + \Delta_{i-1} = \sum_{j \in [n]} L_{j,i} \times A_{j,i}$. This implies that all $A_{i,j}$ for $j \in [n]$ are chosen uniformly at random.² For the last column of the matrix we require that $\Delta_n + \Delta_{n-1} = \Delta_{n-1} = \sum_{j \in [n]} L_{j,n} \times A_{j,n}$, which guarantees that $\langle \hat{\mathbf{A}}, \hat{\mathbf{L}} \rangle = 0$ as required by the protocol. It remains to show that the choice of the $A_{i,j}$ values produced by the simulator is consistent with the simulator's choice of the values β_i .

To this end observe that after a probing attack at most $u \leq d$ columns of the matrix $\hat{\mathbf{A}}$ have been assigned, i.e., the values in $n - d$ columns remain

¹ For instance, suppose that $u = d - 3$ and $v = 3$, then we add $3d - 9$ elements to the set \mathcal{I} .

² At this step we also require that $L_{i,j} \neq 0$ as required by our scheme.

un-assigned. As $\beta_i = a + \sum_j A_{i,j}$ for some fixed value $a = Q_i \times L_i \times R$ the value β_i is distributed uniformly at random even given all values Δ_i and all values $A_{i,j}$ that haven been assigned previously. Notice that this is the case since (a) there is at least one column in the matrix $\hat{\mathbf{A}}$ that has not been assigned yet, and (b) the scheme never computes $\sum_j A_{1,j}$ as β_1 is not needed for the computation. More precisely, let j^* be the column that is unassigned after the assignment of the values above, then each β_i is perfectly hidden by A_{i,j^*} and we can choose A_{1,j^*} in such a way that the column is consistent with Δ_{j^*} . This concludes the analysis of Step C2.

Step C3: To answer the queries on Q_i, R_j and $Q_i \times R_j$ we use the values given in $\{Q_i\}_{i \in \mathcal{I}}$ and $\{R_i\}_{i \in \mathcal{I}}$. To argue that this gives us the right distribution (independent of Q and R), we first recall that by the last claim $|\mathcal{I}| \leq 2d$. Notice that these values will always be consistent with the previously assigned values since each β_i is still blinded by at least one un-assigned value A_{i,j^*} . Moreover, probes of this form (by itself) do not reveal any additional information about $\hat{\mathbf{A}}$.³

Step C4: We can ignore the values sampled in Step C4a as these values remain un-assigned, i.e., they are never directly probed nor are they used in the computation of other probes. The values sampled in Step C4b can be computed from the values that have been assigned previously (since (i, j) was in \mathcal{U} and \mathcal{T} which implies that $i, j \in \mathcal{I}$), and, hence will not affect the joint distribution. Notice that in this step we will always only fix a subset of the $B_{i,j}$ elements in the i -th row, because there are at most $u \leq d$ probes in Step A2 (which lead to adding tuples (i, j) to \mathcal{T}). The remaining values for the i -th row are chosen as defined in Step C4c. That is, in the simulation we choose these values uniformly at random such that $\beta_i = \sum_j L_{i,j} \times B_{i,j}$ taking into account the previously assigned values for $B_{i,j}$ from Step C4b. By the requirement given in Step C4c our choice of $B_{i,j}$ is consistent with the choice of β_i .

It remains to argue why the simulator's choice is also consistent with the matrix $\hat{\mathbf{A}}$ as sampled in Step C2 and with $\Delta_1, \dots, \Delta_n$. To this end, observe that the simulator only samples $B_{i,j}$ according to Step C4c if $A_{i,j}$ has not been revealed previously (i.e., it was in a column of $\hat{\mathbf{A}}$ that has never been probed). Hence, indeed $B_{i,j}$ is uniformly distributed (since it is blinded by the random and unknown value $A_{i,j}$). Finally, it remains to argue that the choice of $B_{i,j}$ is also consistent with the choice of Δ_j from Step B. Recall that for values $B_{i,j}$ for which $(i, j) \in \mathcal{U}$, but not in \mathcal{T} this implies that the j -th row has not been queried. Moreover, we know since $v \leq d$ there are at least $n - v > d$ rows that have not been probed. Hence, there exists some $B_{k,j}$ and $A_{k,j}$ that have not been assigned during the experiment (i.e., they belong to Step C4a). Such values $A_{k,j}$ were never used in the experiment and can always be chosen such that the total sum is consistent with Δ_j .

The above description concludes the proof. □

³ They can, however, reveal information about $B_{i,j}$ as we will see in the next step.

Putting together the above two claims we obtain the statement of the lemma. \square

This completes the analysis of the security of individual masked operations. In the next section we briefly argue about security of masked composed algorithms.

5.2 Security of General Masked Computation

In Section 5.1 we showed that an adversary that probes up to d intermediate values in the computation of the basic masked operation will not be able to learn anything about the underlying sensitive information. We are now interested in what happens to the security when multiple of such operations are combined to carry out some more complicated computation such as the AES encryption algorithm. In other words, we let the adversary learn d intermediate values of the computation carried out by the masked AES algorithm (or any other masked algorithm). Notice that this in particular requires that, e.g., learning d_1 values in a masked multiplication cannot be exploited together with d_2 values learnt from a consecutive masked squaring algorithm as long as $d_1 + d_2 \leq d$.

Similar to earlier work [21], we only provide an informal analysis of d -probing security of composed masked operations. To this end, observe that both in Lemma 2 and Lemma 3 the simulation only depends on at most $2d$ elements of the outputs of the masked operation. As an example consider the masked multiplication that outputs the vector \mathbf{T} and assume that \mathbf{T} is input for a squaring algorithm. If the adversary probes d_1 intermediate values in the multiplication operation, then it is easy to see that the simulation described in Lemma 3 depends on at most $2d_1$ shares of \mathbf{T} . Moreover, the masked multiplication operation guarantees that even given \mathbf{Q} and \mathbf{R} entirely, the output \mathbf{T} is a uniformly and independently chosen maskings of $Q \times R$. This means that for the simulation of the adjacent squaring algorithm the simulator starts with a masking of which $2d_1$ shares are already known. Since according to Lemma 2 the simulator requires d_2 elements of its inputs to simulate the probes in the squaring operation, the simulator will learn additionally d_2 elements of \mathbf{T} . Since $2d_1 + d_2 < n$ this shows security of the simple composed circuit consisting of a multiplication followed by a masked squaring operation. The above argument can easily be extended to arbitrary complicated masked algorithms consisting of many masked operations.

Another difficulty occurs when the adversary can run the masked algorithm multiple times and in each execution he may observe d intermediate values. For instance, one may think of a masked AES algorithm running with a masked key K . Notice that this setting is different from the setting of composed masked computation described above since now the adversary can observe qd intermediate values, where q is the number of executions that the masked algorithm is run. As in earlier work [7, 21] the problem of a continuous probing adversary, i.e., an adversary that learns up to d intermediate values in each execution of the masked AES algorithm, can be addressed by a key refresh algorithm. If (\mathbf{L}, \mathbf{R}) denotes the masking of a key byte of the AES, then the masking of this key byte can be refreshed by running the Algorithm 3 n times consecutively. In other words, the

key refresh algorithm takes as input $\mathbf{R}_0 := \mathbf{R}$ and for $i = 1, \dots, n$ proceeds as follows: Compute $\mathbf{R}_i = \text{IPRefresh}_{\mathbf{L}}(\mathbf{R}_{i-1})$ and output $\mathbf{R}' = \mathbf{R}_n$. Clearly, since we execute $\text{IPRefresh}_{\mathbf{L}}$ n times and the adversary can probe only $d < n$ intermediate values there must exist at least one execution of $\text{IPRefresh}_{\mathbf{L}}(\mathbf{R}_{i-1})$ that does not leak at all. Hence the mask of \mathbf{R} is completely refreshed. This enables us to transform our result to a continuous probing adversary without any additional loss, when in each execution the adversary can learn up to d values (where $2d < n$).

We notice that the fact that $\text{IPRefresh}_{\mathbf{L}}$ is repeated multiple times to refresh the masking of the key is not only an artefact of the security proof. In fact, it is easy to show that for natural implementations of $\text{IPRefresh}_{\mathbf{L}}$ the scheme becomes insecure. To illustrate this, we present a simple attack against a more efficient key refresh algorithm that executes $\text{IPRefresh}_{\mathbf{L}}$ only a single time. For simplicity, let us assume that $\mathbf{L} = (1, \dots, 1)$, i.e., the vector is the all-1 vector. Notice that in this case the inner product masking function is identical to the Boolean masking used, e.g., in [33]. Let us assume that \mathbf{A} is sampled in $\text{IPRefresh}_{\mathbf{L}}$ in the following way:

1. Let $t_0 = 0$. For $i = 1, \dots, n - 1$ repeat the following:
 - (a) Sample A_i uniformly at random in \mathcal{K}
 - (b) Compute $t_i = t_{i-1} + A_i$.
2. Set $A_n = t_n$ and output $\mathbf{A} = (A_1, \dots, A_n)$.

Consider a masked implementation of the AES that at the end of the execution refreshes its key shares by applying a single execution of the $\text{IPRefresh}_{\mathbf{L}}$ algorithm. We now describe an attack that allows to recover the key with only 2 probes in each execution of the masked AES implementation. Notice that we consider the *full probing model* [7], where the adversary can move its probes between consecutive rounds of execution. Our attack does not apply in the restricted probing model [7]. We denote by \mathbf{K}^i the masking of the key k at the beginning of the i -th round, i.e., for all i we have $\langle \mathbf{L}, \mathbf{K}^i \rangle = k$. We have \mathbf{K}^0 being the initially shared key. Moreover, we denote by \mathbf{A}^i the vector that is used in the i -th execution of the masked AES implementation for refreshing, i.e., $\mathbf{K}^i = \mathbf{K}^{i-1} + \mathbf{A}^i$.

1. *First execution of the masked AES:* Probe K_1^0 and A_1^0 . Notice that this allows us to compute K_1^1 .
2. *In the i -th execution of the masked AES:* Probe K_i^i and t_{i-1}^i .

We will now describe how to compute k from the above described probes. Suppose at the beginning of the i -th round (i.e., before carrying out the probes in this round) the adversary knows $\sum_{j=1}^{i-1} K_j^{i-1}$. We show how with the probes described above he can compute $\sum_{j=1}^i K_j^i$. To this end, notice that:

$$\sum_{j=1}^i K_j^i = \sum_{j=1}^{i-1} K_j^{i-1} + t_{i-1}^i + K_i^i,$$

where the second and the third term the adversary knows from the probes in the i -th round and the first term he knows by assumption. Hence, by induction it is easy to argue that the above attack allows to recover the secret key k .

6 Performance Evaluation

We have applied our masking scheme to protect a software implementation of AES-128 encryption. For illustrative purposes we have opted to develop two implementations employing $n = 2$ and $n = 3$ shares, respectively. This choice not only enables a better comparison with other higher-order schemes, but also allows us to gain insight into how the performance scales with an increasing number of shares.

Our target platform is a legacy AVR ATMega163 microcontroller. This device has an 8-bit architecture and offers 32 general purpose registers, 1024 bytes of internal SRAM and 16 KBytes of Flash memory. Our implementation is aimed for speed. To this end, we have written all operations in assembly code and made use of lookup tables whenever possible.

The lowest implementation layer corresponds to arithmetic in the field \mathbb{F}_{2^8} . Field addition is very efficient, as it can be performed in one clock cycle via the native XOR instruction. Field multiplication on the other hand is not part of the AVR instruction set, and we opt to implement it using `log` and `alog` tables [38]. Because this method contains a conditional statement, i.e. check if any of the operands equals zero, realizing it with a constant flow of instructions requires in our implementation 22 cycles. Field squaring - as well as raisings to the power of four and sixteen - are implemented by means of lookup tables. Our platform does not have internal support for generating random numbers, as opposed to e.g. JavaCard smart cards. For the sake of completeness and testing, random numbers are provided externally and stored in memory.

The public parameters \mathbf{L} and $\hat{\mathbf{L}}$ are initialized at setup time and kept constant for each execution of the cipher. Consequently, they are hardcoded in Flash memory. Note that for $n = 2$, there exist 2^8 possible vectors \mathbf{A} orthogonal to \mathbf{L} satisfying $\langle \mathbf{A}, \mathbf{L} \rangle = 0$. These vectors can be as well precomputed during initialization and stored in Flash memory as a look-up table T satisfying $T(A_2) = A_1 = L_2 \times A_2$. During `IPRefreshL`, a value A_2 is picked at random and the corresponding value A_1 is looked up as $T(A_2)$. This allows to improve the efficiency of `IPRefreshL` at the cost of storing 256 bytes in Flash memory.

The main difficulty of applying our masking scheme (and any other) to AES consists in efficiently masking its nonlinear part, i.e. the `SubBytes` transformation. In software contexts it is common to implement this transformation by means of a lookup table. While there exist techniques in the literature to protect table lookups at higher-order, e.g. [7], these are rather costly in terms of performance and storage. Alternatively, one can compute the full `SubBytes` step by using the following equation over \mathbb{F}_{2^8} for a given input state byte X :

$$\begin{aligned} \text{SubBytes}[X] = & \{05\} \times X^{254} + \{09\} \times X^{253} + \{f9\} \times X^{251} + \{25\} \times X^{247} + \\ & \{f4\} \times X^{239} + X^{223} + \{b5\} \times X^{191} + \{8f\} \times X^{127} + \{63\}. \end{aligned}$$

This equation involves multiple field operations (particularly costly multiplications) in order to calculate the different powers of X . It is therefore not very suitable if one aims for an efficient implementation. We therefore follow a different path as already done in [1], i.e. we carry out both steps of the **SubBytes** transformation (inverse and affine transformation) separately. Specifically, we first compute the field inverse by using the following power function:

$$\text{Inverse}[X] = X^{254} = ((X^2 \times X)^4 \times (X^2 \times X))^{16} \times (X^2 \times X)^4 \times X^2.$$

As discussed in [33], this equation can be efficiently carried out with only 4 multiplications and 7 squarings. For the affine transformation (linear only in \mathbb{F}_2) we employ the following equation over \mathbb{F}_{2^8} :

$$\text{AffTrans}[X] = \{05\} \times X^{128} + \{09\} \times X^{64} + \{f9\} \times X^{32} + \{25\} \times X^{16} + \\ \{f4\} \times X^8 + \{01\} \times X^4 + \{b5\} \times X^2 + \{8f\} \times X + \{63\},$$

requiring 7 squarings, 8 additions, and 7 multiplications with a constant.

The **MixCol** transformation operates on the AES state column-by-column. In particular, each of the bytes in the $0 \leq j \leq 3$ columns is replaced as:

$$\begin{aligned} s'_{0,j} &= \{02\} \times s_{0,j} + \{03\} \times s_{1,j} + s_{2,j} + s_{3,j} \\ s'_{1,j} &= s_{0,j} + \{02\} \times s_{1,j} + \{03\} \times s_{2,j} + s_{3,j} \\ s'_{2,j} &= s_{0,j} + s_{1,j} \times \{02\} + s_{2,j} + \{03\} \times s_{3,j} \\ s'_{3,j} &= \{03\} \times s_{0,j} + s_{1,j} + s_{2,j} + \{02\} \times s_{3,j}. \end{aligned}$$

From these equations it follows that this step can be implemented using a total of 12 masked additions and 8 masked multiplications by a constant, for each column. In [10], the authors of AES suggest a more efficient way to compute the **MixCol** step by using the so-called **xtime** tables. Such technique takes advantage of the fact that field addition is more efficient than field multiplication in general purpose processors. Due to this, they suggest an alternative approach that requires 15 additions and 4 multiplications by 02, which can be simply performed as table lookups. We employ this technique to compute the **MixCol** transformation.

The performance of our protected AES-128 implementation is given in Table 2 for $n = 2$ and $n = 3$ secret shares. We have also implemented Boolean masking as proposed in [33] with the same number of secret shares. In order to enable a fair comparison, all implementations are developed on the same platform and follow the same optimization strategy. Finally, we also provide the results given in [1] for the original IP masking using $n = 4$ secret shares. Recall that the original IP masking was developed to provide security order $d = 1$ for $n = 4$, the same as our new scheme and Boolean masking for $n = 3$.

The performance results presented in Table 2 are given in clock cycles. The rightmost column shows the execution time of a full AES-128 encryption including key schedule, while the other columns depict the performances achieved for each AES building block.

Table 2. Performance evaluation (in clock cycles) of protected AES implementations. This work and Boolean with $n = 2$ secret shares (top); this work and Boolean with $n = 3$ secret shares (middle); original IP masking with $n = 4$ secret shares (bottom).

		AddKey	SubBytes	ShiftRows	MixCol	NextSubKey	Full AES
$n = 2$	this work	364	28 810	100	465	7 416	373 544
	Boolean	364	7 850	100	465	2 175	110 569
$n = 3$	this work	476	63 354	150	678	16 148	812 303
	Boolean	476	17 034	150	678	4 568	230 221
$n = 4$	original IP	8 796	117 760	200	27 468	44 437	1 912 000

The improvement with respect to the original IP masking is clear from the results. The overall execution time is reduced by more than a factor 2. This efficiency gain is due to our tweaks in the type of masking leading to an improved construction. In fact, almost all newly proposed operations in the masked domain involve significantly less field operations than in [1]. On the implementation side, this reduction in complexity enables a better usage of the register file, i.e. the number of memory accesses to load/store intermediate results can be significantly reduced.

When compared to Boolean masking for the same security level, our scheme still performs slower. The difference is not entirely due to the operations in the masked domain, but rather to the way the AES affine transformation is defined. Because this operation is linear in \mathbb{F}_2 , protecting it with Boolean masking requires only a matrix multiplication (table lookup) followed by an XOR operation. In contrast, and due to the higher algebraic complexity of the inner product construction, our implementation needs to compute the more complex formula defined over \mathbb{F}_{2^8} . Despite this limitation, our results manage to bridge the gap from approximately a factor 10 to a factor 4. This result makes our proposal an interesting alternative to secure implementations at higher orders.

7 Discussion

In this section we discuss further relevant properties of our improved IP masking scheme and touch on ideas for future work.

Similarities and Differences with Polynomial Masking. A direct consequence of our tweaks is that some characteristics of our construction become closer to those of polynomial masking. In particular, variables in both schemes are encoded using $2n$ shares (L_i, R_i) and (α_i, S_i) respectively, with n shares being public and n shares being secret. The decoding sequences of a masked variable S follow the same pattern of operations, e.g. $S = \langle \mathbf{L}, \mathbf{R} \rangle = \sum_i L_i \times R_i$ and $S = \sum_i \beta_i \times S_i$. We note, however, that the latter is not a consequence of our

simplifications. The same sequence was already used in [1]. In contrast, a direct effect of our tweaks is that the information leakage of encoded secrets using the mutual information as figure of merit becomes comparable.

Despite these high-level similarities, the low-level constructions proposed in this work to carry out operations in the masked domain are different from those proposed in [18, 29]. The only exception is the addition algorithm, in which the secret shares R_i (S_i , respectively) of the input operands are added element-wise to obtain the secret shares of the output. We note that this is also the same for other constructions based on e.g. Boolean masking. The most notable differences can be found in the steps followed to compute the product of two masked variables. In addition, the asymptotic complexity of our multiplication algorithm is $\mathcal{O}(n^2)$ rather than $\mathcal{O}(n^3)$ as presented in [18, 29] or $\tilde{\mathcal{O}}(n^2)$ as described in [8]. Asymptotic improvements are possible using more efficient protocols from multiparty computation – in particular, techniques from multiparty computation using packed secret sharing [14, 20], which results overall in quasi-linear complexity (for large and parallelizable computation).

Finally, we recall that the very nature of both approaches is different. Polynomial masking employs secure multi-party computation techniques [2] and Shamir’s secret-sharing [34], while our construction is inspired by work on leakage resilient cryptography [12]. This difference is for instance prominently reflected in the procedures to mask a variable. In polynomial masking the secret shares are obtained by polynomial evaluation $S_i = P_S(\alpha_i)$ of the public shares α_i , which is different from the procedure described in IPSetup_n . Another difference is that the public parameters α_i of polynomial masking must be both *distinct* and nonzero, while for IP masking only the latter requirement applies, i.e. several L_i can have the same value.

Bounded Leakage Model. We notice that we do not prove the security of our construction in the bounded independent leakage model as done in the work of Balasch *et al.* [1]. Instead, the goal of the current work is to develop an *efficient* higher-order masking scheme that exhibits higher algebraic complexity than Boolean masking and prove its security in the ISW probing model. Note that it is still possible to provide a scheme secure in the independent leakage model even if the vector \mathbf{L} is public but random. The technical reason for this is that the inner product is a strong extractor, i.e. security holds *even* if one part is revealed completely.

The only requirement we need is that the leakage functions are chosen *a priori* and independently of \mathbf{L} , which allows us to rely on the fact that the inner product function over finite fields is a strong extractor [31].⁴ While this may slightly improve the bounds, for small field size, we would still require a large number of shares which may be unrealistic for practical settings. In addition, we would have to make slight changes to the construction, e.g. to our optimized

⁴ We notice that a similar non-adaptive leakage model was considered in [13, 39]. The attack presented in [13] against [39] would not apply in our case since masking schemes are randomized while the stream cipher construction of [39] is deterministic.

squaring algorithm, to prevent simultaneous access to the two halves (\mathbf{L}, \mathbf{R}) of the IP encoding (which becomes insecure under bounded independent leakages).

Resistance Against Glitches. The original scheme in [1] achieves provable security in the presence of glitches *only* when the security parameter n is large. In fact, the proof for glitch resistance follows directly from security in the bounded independent leakage model. However, the construction in this work is proven secure in the probing model, which *does not* automatically imply glitch resistance. Hence we do not claim that our construction is provable secure in the presence of glitches, in contrast to e.g. the polynomial masking scheme by Prouff and Roche [29] and the Threshold Implementation scheme by Nikova et al. [3, 26].

Future Work. An interesting question for future work is if the ideas from [9] can be applied in order to gain a factor 2, i.e. it may be feasible to achieve security against d probes when $d = n - 1$ in the restricted model.

Acknowledgments. Benedikt Gierlichs is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO). This work has been funded in parts the Research Council KU Leuven: GOA TENSE (GOA/11/007). Sebastian Faust received funding from the Marie Curie IEF/FP7 project GAPS, grant number: 626467.

We thank Stefan Dziembowski for many useful discussions on the inner product encoding and the anonymous reviewers of Eurocrypt 2015 for helpful feedback that helped to improve the presentation of our work.

References

1. Balasch, J., Faust, S., Gierlichs, B., Verbauwheide, I.: Theory and Practice of a Leakage Resilient Masking Scheme. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 758–775. Springer, Heidelberg (2012)
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Symposium on Theory of Computing, STOC 1988, pp. 1–10. ACM (1988)
3. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-Order Threshold Implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014)
4. Carlet, C., Goubin, L., Prouff, E., Quisquater, M., Rivain, M.: Higher-Order Masking Schemes for S-Boxes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 366–384. Springer, Heidelberg (2012)
5. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)
6. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2002)
7. Coron, J.-S.: Higher Order Masking of Look-Up Tables. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 441–458. Springer, Heidelberg (2014)

8. Coron, J.-S., Prouff, E., Roche, T.: On the Use of Shamir's Secret Sharing against Side-Channel Analysis. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 77–90. Springer, Heidelberg (2013)
9. Coron, J.-S., Prouff, E., Rivain, M., Roche, T.: Higher-Order Side Channel Security and Mask Refreshing. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 410–424. Springer, Heidelberg (2014)
10. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002)
11. Duc, A., Dziembowski, S., Faust, S.: Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014)
12. Dziembowski, S., Faust, S.: Leakage-Resilient Circuits without Computational Assumptions. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 230–247. Springer, Heidelberg (2012)
13. Faust, S., Pietrzak, K., Schipper, J.: Practical Leakage-Resilient Symmetric Cryptography. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 213–232. Springer, Heidelberg (2012)
14. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: Kosaraju, S.R., Fellows, M., Wigderson, A., Ellis, J.A. (eds.) Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, Victoria, British Columbia, Canada, May 4–6, pp. 699–710. ACM (1992)
15. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine Masking against Higher-Order Side Channel Analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 262–280. Springer, Heidelberg (2011)
16. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
17. Genelle, L., Prouff, E., Quisquater, M.: Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 240–255. Springer, Heidelberg (2011)
18. Goubin, L., Martinelli, A.: Protecting AES with Shamir's Secret Sharing Scheme. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 79–94. Springer, Heidelberg (2011)
19. Goubin, L., Patarin, J.: DES and Differential Power Analysis. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999)
20. Grossbo, V., Standaert, F., Faust, S.: Masking vs. multiparty computation: how large is the gap for AES? *J. Cryptographic Engineering* **4**(1), 47–57 (2014)
21. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
22. 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)
23. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
24. Mangard, S., Popp, T., Gammel, B.M.: Side-Channel Leakage of Masked CMOS Gates. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 351–365. Springer, Heidelberg (2005)

25. Moradi, A., Mischke, O.: On the Simplicity of Converting Leakages from Multivariate to Univariate. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 1–20. Springer, Heidelberg (2013)
26. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology* **24**(2), 292–321 (2011)
27. Prouff, E., Rivain, M.: Masking against Side-Channel Attacks: A Formal Security Proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013)
28. Prouff, E., Rivain, M., Roche, T.: On the Practical Security of a Leakage Resilient Masking Scheme. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 169–182. Springer, Heidelberg (2014)
29. Prouff, E., Roche, T.: Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 63–78. Springer, Heidelberg (2011)
30. Quisquater, J.-J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
31. Rao, A.: An Exposition of Bourgain’s 2-Source Extractor. *Electronic Colloquium on Computational Complexity* **ECCC** 14(034) (2007)
32. Reparaz, O., Gierlichs, B., Verbauwhede, I.: Selecting Time Samples for Multivariate DPA Attacks. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 155–174. Springer, Heidelberg (2012)
33. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
34. Shamir, A.: How to Share a Secret. *Communications of the ACM* **22**(11), 612–613 (1979)
35. Standaert, F.-X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
36. Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The World Is Not Enough: Another Look on Second-Order DPA. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 112–129. Springer, Heidelberg (2010)
37. von Willich, M.: A Technique with an Information-Theoretic Basis for Protecting Secret Data from Differential Power Attacks. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 44–62. Springer, Heidelberg (2001)
38. Win, E.D., Bosselaers, A., Vandenberghe, S., Gersem, P.D., Vandewalle, J.: A Fast Software Implementation for Arithmetic Operations in $GF(2^n)$. In: Kim, K., Matsumoto, T., (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 65–76. Springer, Heidelberg (1996)
39. Yu, Y., Standaert, F., Pereira, O., Yung, M.: Practical leakage-resilient pseudorandom generators. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) Computer and Communications Security, CCS 2010, pp. 141–151. ACM (2010)

Fully Homomorphic Encryption I

Fully Homomorphic Encryption over the Integers Revisited

Jung Hee Cheon^{1(✉)} and Damien Stehlé²

¹ SNU, Seoul, Republic of Korea

jhcheon@snu.ac.kr

² ENS de Lyon, Lyon, France

damien.stehle@ens-lyon.fr

Abstract. Two main computational problems serve as security foundations of current fully homomorphic encryption schemes: Regev’s Learning With Errors problem (LWE) and Howgrave-Graham’s Approximate Greatest Common Divisor problem (AGCD). Our first contribution is a reduction from LWE to AGCD. As a second contribution, we describe a new AGCD-based fully homomorphic encryption scheme, which outperforms all prior AGCD-based proposals: its security does not rely on the presumed hardness of the so-called Sparse Subset Sum problem, and the bit-length of a ciphertext is only $\tilde{O}(\lambda)$, where λ refers to the security parameter.

Keywords: Fully homomorphic encryption · Approximate GCD · DGHV · LWE

1 Introduction

Fully homomorphic encryption has been a major focus of interest in cryptography since Gentry’s first proposal of a fully homomorphic encryption scheme [21, 22]. The security of Gentry’s proposal relies on two hardness assumptions: some relatively ad-hoc problem involving lattices arising in algebraic number theory is assumed intractable, as is the Sparse Subset Sum Problem (SSSP), a variant of the subset sum problem in which the subset is constrained to be very small. The efficiency of Gentry’s scheme was later improved [23, 45, 46], but soon two other design approaches were developed. The interest in Gentry’s original design faded, as the latter approaches rely on better understood hardness assumptions and lead to more efficient instantiations.

Chronologically, the first alternative design was proposed by van Dijk, Gentry, Halevi and Vaikuntanathan [20]. They constructed a fully homomorphic scheme whose security relies on the hardness of SSSP as well as that of the Approximate Greatest Common Divisor problem (AGCD). The AGCD problem, introduced by Howgrave-Graham in [28], is to recover a secret integer p from many approximate multiples $q_i \cdot p + r_i$ of p (see Section 2 for a formal

definition). The efficiency of the DGHV scheme has been improved in a series of works [14, 17–19, 29], and recently adapted to devise a graded encoding scheme serving as an approximation to cryptographic multilinear maps [16].

The other main family of fully homomorphic schemes was initiated by Brakerski and Vaikuntanathan, in [9, 10]. They proposed two fully homomorphic schemes with similar designs. One was relying on the hardness of the Learning With Errors problem (LWE) from [40, 41] while the other used the less understood Ring Learning With Errors problem from [31] to gain on the efficiency front. Note that in both cases, the SSSP hardness assumption is not required anymore. A series of subsequent works proposed efficiency and security improvements, as well as implementations [3, 5, 6, 12, 24, 25, 27].

The co-existence of these two main design strategies for fully homomorphic encryption is due to the combination of circumstances. On one hand, it is not known how the underlying hardness assumptions compare: there is no known reduction from AGCD and SSSP to LWE (or its ring variants), and reciprocally. On the other hand, both approaches seem to lead to implementations whose performances are relatively comparable.

Contributions. This work contains two main results that together lead to a better understanding of the relationship between the AGCD-based and LWE-based fully homomorphic encryption schemes.

Our first contribution is a reduction from LWE to a new and quite natural decision variant of AGCD. Informally, the goal is to distinguish between random approximate multiples $q_i p + r_i$ of a random p and integers uniformly chosen in an interval, with non-negligible distinguishing advantage and non-negligible probability over the choice of p . This AGCD variant is clearly no easier than the search variant considered in [20]. Our reduction implies that for certain distributions for p , the r_i 's and the q_i 's, AGCD is no easier than LWE. It may be combined with Regev's quantum reduction [40, 41] from the approximate variant of the shortest independent vectors problem (SIVP_γ) to LWE. Concretely, if we assume that SIVP_γ in dimension n with $\gamma = \text{poly}(n)$ is exponentially hard to solve (quantumly) with respect to n , which is compatible with the state of the art algorithms for SIVP (see [34]), then AGCD is also exponentially hard to solve, even for bit-sizes of p, r_i, q_i that are quasi-linear in n .

Our second contribution is a fully homomorphic encryption scheme with security based on the hardness of our AGCD variant. In particular, the security does not rely on the presumed hardness of SSSP.¹ The scheme is a variant of the DGHV encryption scheme that embeds the plaintext message in the most significant bit modulo p of an AGCD sample: a ciphertext c corresponding to a plaintext m is of the form $c = qp + \lfloor p/2 \rfloor m + r$. Parameters may be set so that security relies on the quantum hardness of SIVP_γ in dimension n with $\gamma = n^{O(\log n)}$, while the secret key, public/evaluation key and ciphertext expansion remain bounded as $\tilde{O}(\lambda), \tilde{O}(\lambda^3)$ and $\tilde{O}(\lambda)$, where λ is such that all known attacks require time $2^{\Omega(\lambda)}$.

¹ We still require a circular security assumption, like all known fully homomorphic encryption schemes.

Compared to DGHV, we obtain improved asymptotic efficiency and security solely relying on the hardness of LWE. The security and performance are quite similar to those of Brakerski's LWE-based scheme [5]. This is no coincidence, as both contributions build upon ideas from Brakerski's work.

Technical Overview. The reduction from LWE to AGCD relies on several sub-reductions. We use the dimension-modulus trade-off for LWE from [8] and start from 1-dimensional LWE with an exponential modulus q . Informally, the goal is to distinguish from uniform the distribution $(a, a \cdot s + e) \in (\frac{1}{q}\mathbb{Z})/\mathbb{Z} \times \mathbb{R}/\mathbb{Z}$ with a uniform, e small and s an integer. We reduce this variant of LWE to another one where a is instead uniformly sampled in \mathbb{R}/\mathbb{Z} . The computational irrelevance of the discretization parameter q is implicit in [5, 8]: we go one small step further by simply removing it. We then reduce this one-dimensional scale-invariant variant of LWE to the problem considered by Regev in [38] and inspired from [1]. The problem consists in distinguishing from uniform samples of the form $(k + e)/s \in \mathbb{R}/\mathbb{Z}$, where k is uniformly sampled in $[0, s)$ and e is a small noise. A converse reduction was sketched in the appendix of [42], and our reduction was sketched by Oded Regev in a private communication [43]. We formalize the latter reduction. Our chain of reductions improves over the result of [38] in that for comparable hardness assumptions our reduction allows to take a bitsize for s that is the square root of that allowed by [38]. Finally, we scale and re-discretize samples $(k + e)/s \in \mathbb{R}/\mathbb{Z}$ to obtain a reduction from the latter problem to a decision variant of AGCD.

Our encryption scheme is inspired from that of [38] and the LWE-based Brakerski's fully homomorphic encryption scheme [5]. It is scale-invariant in the sense that it remains unchanged if we multiply both the secret key p and the ciphertext by the same quantity. It does not use a hidden greatest common divisor that is a square as in the Coron *et al.* scale invariant version of the DGHV scheme [17]. Homomorphic addition comes without extra work. For homomorphic multiplication, we adapt the dimension-reduction technique from [9], that uses (invalid) encryptions of the bits of secret p (we assume that it is safe to publish these data, hence making a circular-security assumption). Finally, we bound the multiplicative depth of the decryption circuit is bounded as $O(\log \lambda)$ where λ refers to the security parameter. As the parameters may be set so that our homomorphic scheme supports this multiplicative depth, it is hence possible to bootstrap it [22], leading to a fully homomorphic encryption scheme. This allows us to circumvent the SSSP hardness assumption made in prior variants of the DGHV encryption scheme.

Finally, we propose a modification of our scheme in which the ciphertext bit-size is reduced. This is achieved by truncating the least significant bits of the ciphertext, which is made possible by the fact that the plaintext is not embedded into these. As a result, the ciphertext size is almost as low as $\gamma - \rho$, where γ is the bit-length of the AGCD samples and ρ is the bit-length of the AGCD noise. We remark that one can additionally use the technique of [19] to compress the public key.

Open Problems. Our results show that the DGHV fully homomorphic encryption scheme [20] can be made to fit into the LWE landscape. The modified scheme asymptotically outperforms DGHV (and all subsequent variants), but its performance only matches Brakerski’s LWE-based scheme [5]. Further, there exist recent LWE-based fully homomorphic encryption schemes with strengthened security [3, 11, 27],² and efficiency can be increased if one relies on the ring variant of LWE problem.

With this state of affairs, it may be tempting to drop the AGCD approach altogether. We prefer a more optimistic interpretation of our work. First, it gives greater confidence into the hardness of AGCD and simplifies AGCD-based encryption. This clearer landscape could serve as a firmer grounding for further developments. The AGCD problem can be seen as another way of expressing LWE: it may turn out to be more convenient for cryptographic design. Finally, the analogy does not seem to be complete: some variants of DGHV rely on a modification of AGCD in which a noiseless multiple of the secret integer p is published [18] (the security of these variants relies on an extra hardness assumption related to factoring). We are not aware of a similar problem in the LWE landscape.

Our scheme is relatively slow (compared to those based on Ring LWE), but several existing techniques could be exploited to accelerate it. For instance, it may be possible to pack more plaintexts into a single ciphertext, similarly to [14]. It may also be possible to refine the bootstrapping step rather than looking at decryption as a generic binary circuit. Finally, our variant with truncated ciphertexts raises the question of taking AGCD instances with small $(\gamma - \rho)$. To thwart attacks based on exhaustive search, we should have $\gamma - \rho \geq \lambda + \Omega(\log \lambda)$. If $\gamma - \rho \approx \lambda + \Omega(\log \lambda)$ turns out to be safe, then the ciphertext bit-sizes of our variant scheme based on truncation can be made quite small.

Road-Map. We describe our LWE to AGCD reduction in Section 2. In Section 3, we describe our AGCD-based scheme, and we show in Section 4 how it may be extended into a fully homomorphic encryption scheme. Section 5 contains a modification of the scheme with smaller ciphertexts.

Notation. We use standard Landau notations. When manipulating reals, we in fact manipulate finite-precision approximations, with polynomially many bits of precision. If x is a real, then $\lfloor x \rfloor$ refers to the nearest integer to x , rounding upwards in case of a tie. The notation \log refers to the base-2 logarithm. We use the notation $(a_i)_{i=1,\dots,k}$ or simply $(a_i)_i$ for a vector (a_1, \dots, a_k) . Given $x, p \in \mathbb{R}$, we let $[x]_p$ denote the unique number in $(-p/2, p/2]$ that is congruent to x modulo p . The notation is extended to vectors $\mathbf{x} \in \mathbb{R}^n$ in the obvious way. We let \mathbb{T} denote the torus \mathbb{R}/\mathbb{Z} . For an integer $q \geq 1$, we let \mathbb{T}_q denote the set $\{0, 1/q, \dots, (q-1)/q\}$ with addition modulo 1.

We use $a \leftarrow A$ to denote the operation of uniformly sampling an element a from a finite set A . When \mathcal{D} is a distribution, the notation $a \leftarrow \mathcal{D}$ refers to

² Note that it may be possible to adapt these techniques to the AGCD framework. A DGHV variant was proposed in appendix of [27].

sampling a according to distribution \mathcal{D} . We recall that the statistical distance between two distributions \mathcal{D}_1 and \mathcal{D}_2 with supports contained in a common measurable set is half the ℓ_1 -norm of their difference.

If X is a set of finite weight, we let $U(X)$ denote the uniform distribution over X . For a parameter $s > 0$, we let D_s denote the (continuous) Gaussian distribution of parameter s , i.e., the law over \mathbb{R} with density function $x \mapsto \exp(-\pi x^2/s^2)/s$. We write $D_{\leq s}$ to refer to a $D_{s'}$ for some $s' \leq s$. If $\Lambda \subseteq \mathbb{R}^n$ is a full-rank lattice, $s > 0$ and $\mathbf{c} \in \mathbb{R}^n$, we let $D_{\Lambda, s, \mathbf{c}}$ denote the (discrete) Gaussian distribution with support Λ and density function $\mathbf{x} \mapsto \exp(-\pi \|\mathbf{x} - \mathbf{c}\|^2/s^2)/C$ with $C = \sum_{\mathbf{x} \in \Lambda} \exp(-\pi \|\mathbf{x} - \mathbf{c}\|^2/s^2)$. When $\mathbf{c} = \mathbf{0}$, we omit the last subscript. We recall a few properties of Gaussians in Appendix A.

We say that a distribution \mathcal{D} over \mathbb{Z}^n is (B, ε) -bounded if $\Pr_{x \leftarrow \mathcal{D}}[\|x\| \leq B] \geq 1 - \varepsilon$. We say that \mathcal{D} is (B, δ, ε) -contained if $\Pr_{x \leftarrow \mathcal{D}}[\|x\| \in [\delta B, B]] \geq 1 - \varepsilon$. For example, for all $\varepsilon \in (0, 1/2)$, the distribution $D_{\mathbb{Z}^n, r}$ is (B, ε) -bounded, with $B = O(r\sqrt{n \ln(n/\varepsilon)})$ (see [32]). If $r = \Omega(\sqrt{\ln(1/\varepsilon)})$, then the distribution $D_{\mathbb{Z}, r}$ is (B, δ, ε) -contained, with $B = O(r\sqrt{\ln(1/\varepsilon)})$ and $\delta = \varepsilon/\sqrt{\ln(1/\varepsilon)}$.

Throughout the paper, we let λ denote the security parameter: all known valid attacks against the cryptographic scheme under scope should require $2^{\Omega(\lambda)}$ bit operations to mount.

2 Hardness of Approximate GCD

We exhibit a reduction from the Learning With Errors problem (LWE) to a variant of the Approximate Greatest Common Divisor problem (AGCD). We first introduce the precise problems under scope.

We will consider the following decision variant of AGCD. The corresponding search variant (consisting in finding the unknown p) is frequent in the literature. There exists a (trivial) reduction from the search variant to the decision variant. Other decision variants of AGCD were considered in [17, 19, 29]. We believe that our decision variant of AGCD is more natural as it is less application-driven.

Definition 1 (AGCD). Let $p, X \geq 1$, and ϕ a distribution over \mathbb{Z} (that can depend on p). We define $A_{X, \phi}^{\text{AGCD}}(p)$ as the distribution over \mathbb{Z} obtained by sampling $q \leftarrow \mathbb{Z} \cap [0, X/p]$ and $r \leftarrow \phi$, and returning $x = q \cdot p + r$.

Let \mathcal{D} be a distribution over $\mathbb{Z} \cap [0, X]$. $\text{AGCD}_{X, \phi}(\mathcal{D})$ consists in distinguishing, given arbitrarily many independent samples, between the uniform distribution over $\mathbb{Z} \cap [0, X]$ and the distribution $A_{X, \phi}^{\text{AGCD}}(p)$ for a fixed $p \leftarrow \mathcal{D}$. We use the notation $\text{AGCD}_{X, \phi}^m(\mathcal{D})$ to emphasize the number of samples m used by the eventual distinguisher.

We say that an algorithm \mathcal{A} is an $(\varepsilon_1, \varepsilon_2)$ -distinguisher for $\text{AGCD}_{X, \phi}(\mathcal{D})$ if, with probability $\geq \varepsilon_2$ over the randomness of $p \leftarrow \mathcal{D}$, its distinguishing advantage between $A_{X, \phi}^{\text{AGCD}}(p)$ and $U(\mathbb{Z} \cap [0, X])$ is $\geq \varepsilon_1$.³

³ We do not explicitly focus on the distinguishing run-times, as our reductions almost preserve run-times.

For $\rho, \eta, \gamma \geq 1$, the (ρ, η, γ) -AGCD problem is AGCD $_{2^\gamma, \phi}^m(\mathcal{D})$ with \mathcal{D} the uniform distribution over η -bit prime integers and ϕ the uniform distribution over $\mathbb{Z} \cap (-2^\rho, 2^\rho)$.

We will not rely on (ρ, η, γ) -AGCD in our constructions, but we recall it for comparison convenience with prior works. First, we do not need to impose that the secret p is prime. In fact, there is no known attack that exploits the factorization of p . Also, if the distribution \mathcal{D} is sufficiently well-behaved, restricting \mathcal{D} to prime integers (e.g., by rejection sampling) would result in a problem that is no easier, as the density of prime numbers is non-negligible. Second, we do not know how to reduce LWE to (ρ, η, γ) -AGCD. In particular, our reduction leads to distributions \mathcal{D} and ϕ that are somewhat more cumbersome. However, from the perspective of cryptographic constructions, they may be used in the exact same manner as their (ρ, η, γ) -AGCD counterparts.

LWE was introduced by Regev [41]. We use the variant from [8].

Definition 2 (LWE). Let $n, q \geq 1$, $s \in \mathbb{Z}^n$ and ϕ a distribution over \mathbb{R} . We define $A_{q, \phi}^{\text{LWE}}(s)$ as the distribution over $\mathbb{T}_q^n \times \mathbb{T}$ obtained by sampling $a \leftarrow \mathbb{T}_q^n$ and $e \leftarrow \phi$, and returning $(a, \langle a, s \rangle + e)$.

Let \mathcal{D} be a distribution over \mathbb{Z}^n . LWE $_{n, q, \phi}(\mathcal{D})$ consists in distinguishing, given arbitrarily many independent samples, between $U(\mathbb{T}_q^n \times \mathbb{T})$ and $A_{q, \phi}^{\text{LWE}}(s)$ for a fixed $s \leftarrow \mathcal{D}$.

In [41], Regev described a quantum reduction from several standard (worst-case) problems over n -dimensional Euclidean lattices to LWE $_{n, p, D_{\leq \alpha}}(U((\mathbb{Z} \cap [0, p))^n))$, where the modulus p may be chosen as a polynomial in n and the parameter α may be set as $\text{poly}(n)/\gamma$ with γ referring to the approximation factor of the considered lattice problem. The reduction assumes that $\alpha \geq \Omega(\sqrt{n}/q)$. Regev's reduction was partly dequantized in [36] and [8]. Further, a modulus-dimension trade-off was exhibited in [8]: in particular, if $q = \Omega(p^n)$ and $\alpha \leq \text{poly}(n)\beta$, then LWE $_{1, q, D_{\leq \alpha}}(U(\mathbb{Z} \cap [0, q]))$ is no easier than LWE $_{n, p, D_{\leq \beta}}(U((\mathbb{Z} \cap [0, p))^n))$.

In [4], Applebaum *et al* gave an LWE self-reduction from secret distribution $U((\mathbb{Z} \cap [0, p))^n)$ to secret distribution $D_{\mathbb{Z}^n, O(\alpha p)}$ which reduces the distinguishing advantage from ε to $\Omega(\varepsilon)$, if $\alpha \geq \Omega(\sqrt{\ln(n/\varepsilon)}/p)$.

The main result of this section is the following.

Theorem 1. Let $\alpha, \beta \in (0, 1)$, $X, B, m, q \geq 1$, and \mathcal{D} a distribution over \mathbb{Z} . Assume that there exists an $(\varepsilon_1, \varepsilon_2)$ -distinguisher for AGCD $_{X, \lfloor D_{\leq \alpha} \rfloor}^m(|X/\mathcal{D}|)$. If \mathcal{D} is $(B, \delta, \varepsilon_2/2)$ -contained, $q \geq \Omega(\sqrt{\ln(m/\varepsilon_1)}B/\beta)$, $X \geq \Omega(mB^2/(\beta\varepsilon_1))$ and $\beta \leq O(\alpha\delta B/X)$, then there exists an $(\Omega(\varepsilon_1), \Omega(\varepsilon_2\delta\beta/\sqrt{\ln(m/\varepsilon_1)}))$ -distinguisher for LWE $_{1, q, D_{\leq \beta}}^m(\mathcal{D})$.

Setting Parameters in AGCD. We discuss a possible choice of secure parameters for AGCD.

Recall that there exists a (quantum) reduction from λ -dimensional lattice problems with approximation factors $\lambda^{\tilde{O}(1)}$, to LWE $_{1, q, D_{\leq \beta'}}(D_{\mathbb{Z}, \sigma})$, for $q = 2^{\tilde{O}(\lambda)}$,

$\beta' = \lambda^{-\tilde{\Omega}(1)}$ and $\sigma = O(\beta'q)$. We can hence reasonably assume that for these parameters, the time required to solve $\text{LWE}_{1,q,D_{\leq \beta'}}(D_{\mathbb{Z},\sigma})$ is $2^{\tilde{\Omega}(\lambda)}$, even for $\varepsilon_1, \varepsilon_2$ as small as $2^{-\tilde{\Omega}(\lambda)}$.

We set $\beta = \sqrt{\lambda}\beta'$. As we can publicly increase the Gaussian noise of the LWE samples, $\text{LWE}_{1,q,D_{\leq \beta}}(D_{\mathbb{Z},\sigma})$ is no easier than the latter variant. Now, Theorem 1 depends quite importantly on the potential smallness of samples from \mathcal{D} . We avoid such small values by rejection sampling. We define \mathcal{D} as follows: Sample a fresh $s \leftarrow D_{\mathbb{Z},O(\sigma)}$ until $s \in (\sigma/2, 2\sigma)$. As a result, we can set $B = O(\sigma)$ and $\delta = \Omega(1)$. The condition on q in Theorem 1 is fulfilled. If we set $X = 2^\lambda\sigma^2$ and $\alpha = \Omega(\beta X/\sigma) \approx \beta 2^\lambda\sigma$, then all conditions are fulfilled, guaranteeing exponential hardness of $\text{AGCD}_{X,\lfloor D_{\leq \alpha} \rfloor}^m(\mathcal{D}_\sigma^*)$, with $\mathcal{D}_\sigma^* = \lfloor X/\mathcal{D} \rfloor$.

To ease comparison with prior works, we define

$$\gamma = \log X, \quad \eta = \log X - \log \sigma, \quad \text{and } \rho = \log \alpha + (\log \lambda)/2.$$

The bit-size of each AGCD sample $pq + r$ is $\approx \gamma$, the bit-size of the AGCD secret p is $\approx \eta$, and the bit-size of each noise term r is bounded by ρ , with probability exponentially close to 1 (in the analysis of the primitives, we will assume that each fresh noise r has magnitude $\leq 2^\rho$, hence forgetting about the unlikely event that one of the noises is bigger). With our choices of X and α , we have: $\gamma \approx \lambda + 2\log \sigma$, $\eta \approx \lambda + \log \sigma$ and $\rho \approx \eta + \log(\sqrt{\lambda}\beta)$.

Proof Overview. The proof of Theorem 1 consists of three sub-reductions. We first show that LWE is essentially equivalent to a variant of LWE that does not involve any discretization parameter q . That variant, which we name scale-invariant LWE (SILWE), is implicit in [5, 8]. We then show that SILWE is essentially equivalent to the problem studied in [39] (and inspired from [1]), which we name zero-dimensional LWE (ZDLWE). Finally, the third sub-reduction is from ZDLWE to AGCD.

In Appendix B, we give converse reductions for each one of the three sub-reductions. This implies that from the hardness viewpoint, AGCD and LWE are quite closely related.

2.1 Scale-Invariant LWE

We consider the following LWE variant, in which the modulus q does not play a role anymore.

Definition 3 (Scale-Invariant LWE). Let $n \geq 1$, $\mathbf{s} \in \mathbb{Z}^n$ and ϕ a distribution over \mathbb{R} . We define $A_\phi^{\text{SILWE}}(\mathbf{s})$ as the distribution over $\mathbb{T}^n \times \mathbb{T}$ obtained by sampling $\mathbf{a} \leftarrow \mathbb{T}^n$ and $e \leftarrow \phi$, and returning $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$.

Let \mathcal{D} be a distribution over \mathbb{Z}^n . $\text{SILWE}_{n,\phi}(\mathcal{D})$ consists in distinguishing, given arbitrarily many independent samples, between $U(\mathbb{T}^n \times \mathbb{T})$ and $A_\phi^{\text{SILWE}}(\mathbf{s})$ for a fixed $\mathbf{s} \leftarrow \mathcal{D}$.

Lemma 1. Let $\alpha, \beta \in (0, 1)$, $m, n, q, B \geq 1$ and \mathcal{D} a distribution over \mathbb{Z}^n . Assume that there exists an $(\varepsilon_1, \varepsilon_2)$ -distinguisher for $\text{SILWE}_{n,D_{\leq \alpha}}^m(\mathcal{D})$. If \mathcal{D} is

$(B, \varepsilon_2/2)$ -bounded, $q \geq \Omega(\sqrt{\ln(mn/\varepsilon_1)}B/\beta)$ and $\beta \leq O(\alpha)$, then there exists an $(\Omega(\varepsilon_1), \Omega(\varepsilon_2))$ -distinguisher for $\text{LWE}_{n,q,D_{\leq\beta}}^m(\mathcal{D})$.

Proof. We map each input sample (\mathbf{a}, b) for $\text{LWE}_{n,q,D_{\leq\beta}}$ to an input sample (\mathbf{a}', b') for $\text{SILWE}_{n,D_{\leq\alpha}}$, as follows: Sample $\mathbf{f} \leftarrow D_r^n$ with $r = \Omega(\sqrt{\ln(mn/\varepsilon_1)})/q$; set $\mathbf{a}' = \mathbf{a} + \mathbf{f}$ and $b' = b$. We show below that, with probability $\geq 1 - \varepsilon_2/2$ over the randomness of $\mathbf{s} \leftarrow \mathcal{D}$, this transformation maps the distributions $U(\mathbb{T}_q^n \times \mathbb{T})$ and $A_{q,D_{\leq\beta}}^{\text{LWE}}(\mathbf{s})$ to distributions within statistical distances $O(\varepsilon_1/m)$ from $U(\mathbb{T}^n \times \mathbb{T})$ and $A_{D_{\leq\alpha}}^{\text{SILWE}}(\mathbf{s})$, respectively.

By Lemma 10, the distribution of $\mathbf{f} \bmod 1$ is within statistical distance $O(\varepsilon_1/m)$ from $U(\mathbb{T}^n)$, and the distribution of \mathbf{f} conditioned on $(\mathbf{f} \bmod 1)$ is $D_{\mathbb{Z}^n/q,r}$. The former implies that \mathbf{a}' is uniformly distributed over \mathbb{T}^n . Now, we consider two cases. If b was uniformly distributed in \mathbb{T} independently of \mathbf{a} , then b' is uniformly distributed in \mathbb{T} independently of \mathbf{a}' . Now, assume that $b = \langle \mathbf{a}, \mathbf{s} \rangle + e$ for some fixed \mathbf{s} and $e \leftarrow D_{\leq\beta}$. Then $b' - \langle \mathbf{a}', \mathbf{s} \rangle = e - \langle \mathbf{f}, \mathbf{s} \rangle$. By Lemma 12, the distribution of $e - \langle \mathbf{f}, \mathbf{s} \rangle$ (conditioned on \mathbf{a}') is within statistical distance $O(\varepsilon_1/m)$ from $D_{\leq\sqrt{\beta^2+\|\mathbf{s}\|^2}r^2}$, assuming that $(1/r^2 + \|\mathbf{s}\|^2/\beta^2)^{-1/2} \geq \Omega(\sqrt{\ln(mn/\varepsilon_1)})/q$. We have $\|\mathbf{s}\| \leq B$ with probability $\geq 1 - \varepsilon_2/2$ (over the randomness of \mathbf{s}). When this is the case, we obtain that $e - \langle \mathbf{f}, \mathbf{s} \rangle$ (conditioned on \mathbf{a}') is within statistical distance $O(\varepsilon_1/m)$ from $D_{\leq\sqrt{\beta^2+\|\mathbf{s}\|^2}r^2}$ (by using the condition on q and the definition of r). Finally, the assumptions on β , B and r ensure that $\sqrt{\beta^2 + \|\mathbf{s}\|^2 r^2} \leq \alpha$. \square

2.2 Zero-Dimensional LWE

We now show that SILWE is essentially equivalent to the problem studied by Regev in [39]. The latter may be viewed as a zero-dimensional variant of LWE, as the provided samples are from \mathbb{T} rather than $\mathbb{T}_q^n \times \mathbb{T}$.

Definition 4 (Zero-Dimensional LWE). Let $s \in \mathbb{Z}$ and ϕ a distribution over \mathbb{R} . We define $A_\phi^{\text{ZDLWE}}(s)$ as the distribution over \mathbb{T} obtained by sampling $k \leftarrow \mathbb{Z} \cap [0, s)$ and $e \leftarrow \phi$, and returning $[(k + e)/s]_1$.

Let \mathcal{D} be a distribution over \mathbb{Z} . $\text{ZDLWE}_\phi(\mathcal{D})$ consists in distinguishing, given arbitrarily many independent samples, between $U(\mathbb{T})$ and $A_\phi^{\text{ZDLWE}}(s)$ for a fixed $s \leftarrow \mathcal{D}$.

The following result and its proof are derived from [43].

Lemma 2. Let $\alpha, \beta \in (0, 1)$, $B \geq 1$ and \mathcal{D} a distribution over \mathbb{Z} . Assume that there exists an $(\varepsilon_1, \varepsilon_2)$ -distinguisher for $\text{ZDLWE}_{D_{\leq\alpha}}^m(\mathcal{D})$. If \mathcal{D} is $(B, \delta, \varepsilon_2/2)$ -contained and $\beta \leq O(\alpha)$, then there exists an $(\Omega(\varepsilon_1), \Omega(\varepsilon_2\delta\alpha/\sqrt{\ln(m/\varepsilon_1)}))$ -distinguisher for $\text{SILWE}_{1,D_{\leq\beta}}^m(\mathcal{D})$.

Proof. We describe a reduction from SILWE to ZDLWE. Let $r = \Theta(\sqrt{\ln(m/\varepsilon_1)})$ (chosen to be able to use Lemma 10) and $\delta' \leq \Theta(\delta\alpha/\sqrt{\ln(m/\varepsilon_1)})$. The reduction

produces a guess s' of the SILWE secret s by sampling $s' \leftarrow B\delta' \cdot (\mathbb{Z} \cap [0, \lceil 1/\delta' \rceil])$; then it maps any input sample (a, b) for $\text{SILWE}_{1, D_{\leq \beta}}$ to an input sample y for $\text{ZDLWE}_{D_{\leq \alpha}}$, by setting $y = [a - b/s']_1$.

This transformation maps $U(\mathbb{T} \times \mathbb{T})$ to $U(\mathbb{T})$. We now show that it maps $A_{D_{\leq \beta}}^{\text{SILWE}}(s)$ to a distribution that is within statistical distance $O(\varepsilon_1/m)$ from $A_{D_{\leq \alpha}}^{\text{ZDLWE}}(s)$, with probability $\Omega(\varepsilon_2\delta')$ over the choice of $s \leftarrow \mathcal{D}$. Thanks to the assumption on \mathcal{D} , we have that $|s| \in [\delta B, B]$, with probability $\geq 1 - \varepsilon_2/2$ over the randomness of s . The success probability of the $\text{ZDLWE}_{D_{\leq \alpha}}$ distinguisher conditioned on that event is $\geq \varepsilon_2/2$. Further, with probability $\geq \Omega(\delta')$ over the choice of s' , we have $|s' - s| \leq B\delta'$. We now assume that $|s| \in [\delta B, B]$, $|s' - s| \leq B\delta'$ and that the $\text{ZDLWE}_{D_{\leq \alpha}}$ distinguisher succeeds. This event has weight $\Omega(\varepsilon_2\delta')$.

By Lemma 10, the distribution of a is within statistical distance $O(\varepsilon_1/m)$ of the distribution obtained by sampling $k \leftarrow \mathbb{Z} \cap [0, s)$ and $f \leftarrow D_r$, and returning $[(k + f)/s]_1$. With these notations, we have $b = f + e \bmod 1$ and $y = k/s + f(1/s - 1/s') - e/s' \bmod 1$. The distribution of $sy - k$ is within statistical distance $O(\varepsilon_1/m)$ of $D_{\leq \alpha'}$ with $\alpha' = ((\beta s/s')^2 + r^2(1 - s/s')^2)^{1/2}$. Thanks to the properties on s and s' , we have $|s/s'| \leq O(1)$ and $|1 - s/s'| \leq O(\delta'/\delta)$. This leads to $\alpha' \leq O(\beta + r\delta'/\delta)$. The condition on β and the choice of δ' ensure that $\alpha' \leq \alpha$. \square

Note that the hardness result obtained here via LWE is stronger than the one from [39]. Indeed, the present approach leads to a (quantum) reduction from standard n -dimensional lattice problems with polynomial approximation factors to ZDLWE with an s of bitsize $\tilde{O}(n)$, whereas [39] leads to an s of bitsize $\tilde{O}(n^2)$. However, the latter reduction is classical rather than quantum.

2.3 Reducing ZDLWE to AGCD

Lemma 3. *Let $\alpha, \beta \in (0, 1)$, $X, B \geq 1$ and \mathcal{D} a distribution over \mathbb{Z} . Assume that there exists an $(\varepsilon_1, \varepsilon_2)$ -distinguisher for $\text{AGCD}_{X, \lfloor D_{\leq \alpha} \rfloor}^m(\lceil X/\mathcal{D} \rceil)$. If \mathcal{D} is $(B, \delta, \varepsilon_2/2)$ -contained, $X \geq \Omega(mB^2/(\alpha\varepsilon_1))$ and $\beta \leq O(\alpha\delta B/X)$, then there exists an $(\Omega(\varepsilon_1), \Omega(\varepsilon_2))$ -distinguisher for $\text{ZDLWE}_{D_{\leq \beta}}^m(\mathcal{D})$.*

Proof. Given an input sample y for $\text{ZDLWE}_{D_{\leq \beta}}$, the reduction produces an input sample x for $\text{AGCD}_{D_{\leq \alpha}}^m$, as follows: Set $x = [\lfloor Xy \rfloor]_X$.

If y is uniformly distributed over \mathbb{T} , then so is x over $\mathbb{Z} \cap [0, X)$. Now, assume that $y = (k+e)/s$ for some fixed s (sampled from \mathcal{D}), $k \leftarrow \mathbb{Z} \cap [0, s)$ and $e \leftarrow D_{\leq \beta}$. Then $x = kp + r - \Delta$ with $p = \lfloor X/s \rfloor$, $r = \lfloor Xe/s \rfloor$ and $\Delta = \lfloor X(k+e)/s \rfloor - k\lfloor X/s \rfloor - r$. We have $|\Delta| \leq 2 + k \leq O(B)$, with probability $\geq 1 - \varepsilon_2/2$ over the choice of $s \leftarrow \mathcal{D}$. The distribution of r is $[D_{\alpha'}]$ for some $\alpha' \in [X\beta/B, X\beta/(\delta B)]$ (where we used the fact that $|s| \geq \delta B$, which holds with high probability over the choice of s , by assumption on \mathcal{D}). We observe that the statistical distance between $[D_{\alpha'}]$ and $[D_{\alpha'}] - \Delta$ is $O(\Delta/\alpha') \leq O(B^2/(X\beta)) \leq O(\varepsilon_1/m)$.

It now suffices to show that the distribution of $k \leftarrow \mathbb{Z} \cap [0, s)$ is statistically close to the uniform distribution over $\mathbb{Z} \cap [0, X/p)$. A simple calculation shows

that the statistical distance between the uniform distributions over these two intervals is $O(|X/p - s|/s)$. By definition of p , we have that $|ps - X| \leq s$ and hence $|X/(ps) - 1| \leq 1/p$. The latter is $O(B/X) \leq O(\varepsilon_1/m)$, thanks to the assumption on X . \square

Theorem 1 is obtained by combining Lemmas 1, 2 and 3.

3 An AGCD-Based Additive Homomorphic Encryption Scheme

In this section, we propose an additive homomorphic encryption (AHE) scheme whose security relies on the hardness of the AGCD problem. This scheme is similar to the DGHV encryption scheme [20], but the plaintext message is embedded into the ciphertext c as the most significant bit of $c \bmod p$ for the secret key p . It may be viewed as an AGCD adaptation of Regev's encryption scheme from [38].

We let ρ denote a bound on the bit-length of the error, η the bit-length of the secret greatest common divisor, and γ the bit-length of an AGCD sample. The parameter τ refers to the number of encryptions of zero contained in the public key.

Parameters. We set parameters such that they satisfy the following constraints.

- $\rho \geq \lambda$, to protect against the brute force attacks on the noise such as [13, 28].
- $\gamma \geq \Omega\left(\frac{\lambda}{\log \lambda}(\eta - \rho)^2\right)$ and $\gamma \leq \eta^2$, to thwart the lattice reduction attacks on AGCD such as the orthogonal lattice attacks [20, 35], Lagarias' simultaneous Diophantine approximation [30] and the Cohn-Heninger attack [15].
- η will be determined later to support correct decryption. For the moment, we only suppose that $\rho < \eta$.
- $\tau = \gamma + 2\lambda + 2$, to be able to use the leftover hash lemma in the security proof (see Subsection 3.3).

Note that there is some discrepancy with the conditions with prior works on AGCD. Part of it stems from the fact that we place ourselves in the context of sub-exponential attackers rather than polynomial-time attackers. Further, once adapted to this attacker setup, the condition corresponding to thwarting lattice attacks is $\gamma \geq \Omega(\lambda\eta^2)$ in prior works. In fact, that condition is too stringent: lattice attacks are thwarted even if our (weaker) condition is satisfied. Moreover, our condition is compatible with the LWE to AGCD reduction when applied to exponentially intractable LWE parameters, as explained in Section 2. This has a significant impact on the asymptotic performance of the scheme, as γ may be set much smaller.

Concretely, we set $\rho = \lambda$, $\eta = \rho + L \log \lambda$ for an $L > 0$ to be chosen to provide desirable functionalities, $\gamma = \Omega(L^2 \lambda \log \lambda)$ and $\tau = \gamma + 2\lambda + 2$. Note that the ciphertext size γ is quasi-linear in λ . Assume one wants to rely on the exponential hardness of lattice problems for approximation factors $n^{O(L)}$ for a small L . First, one has to set $n = \Omega(L\lambda)$. In that case, via the reduction from Section 2, one can set $\sigma = \Omega(L^2 \lambda \log \lambda)$, and $\eta' = cL^2 \lambda \log \lambda$ for some constant c , $\rho' = \eta' - L \log \lambda$, $\gamma' = 2\eta' + \lambda$ and $\tau' = \gamma' + 2\lambda + 2$.

3.1 The Construction

The scheme **AHE** is defined as follows:

AHE.KeyGen(λ). Given a security parameter λ , determine parameters (X, σ, α) and (γ, η, ρ) providing security λ and decryption correctness (see analysis below). We refer to the discussion just after Theorem 1 for the relationship between the two parameter sets. Sample $p \leftarrow \mathcal{D}_\sigma^*$ (of bitsize $\approx \eta$). For $0 \leq i \leq \tau$, sample $x_i \leftarrow A_{X, [D_\alpha]}^{\text{AGCD}}(p)$. Relabel so that x_0 is the largest and x_1 has an odd $\lfloor \frac{x_1}{p} \rfloor$, and restart if we cannot find such an x_1 . Output the secret key $sk = p$ and the public key $pk = (x_0, x_1, \dots, x_\tau)$.

AHE.Enc _{pk} (m). Given a message $m \in \{0, 1\}$, uniformly sample a subset $S \subseteq \{1, 2, \dots, \tau\}$, and output

$$c = \left[\sum_{i \in S} x_i + \left\lfloor \frac{x_1}{2} \right\rfloor m \right]_{x_0}.$$

AHE.Add _{x_0} (c_1, c_2). Given two ciphertexts c_1, c_2 , output $c_{\text{add}} = [c_1 + c_2]_{x_0}$.

AHE.Dec _{sk} (c). Given a ciphertext c , output $m = \left[\left\lfloor \frac{2c}{p} \right\rfloor \right]_2$.

Note that $\left[\lfloor x \rfloor \right]_2$ may not be equal to $\lfloor [x]_2 \rfloor$ for some $x \in \mathbb{R}$. In fact, the latter has value in $\{0, 1, -1\}$ while the former has value in $\{0, 1\}$. However, they are congruent modulo 2.

3.2 Correctness

We analyze the noise growth at encryption and addition, and provide a sufficient condition for decryption correctness.

Lemma 4 (Encryption noise). *Let $(sk = p, pk = (x_0, \dots, x_\tau)) \leftarrow \text{AHE.KeyGen}(\lambda)$ and $c \leftarrow \text{AHE.Enc}_{pk}(m)$ for a message $m \in \{0, 1\}$. Then*

$$c = r + \left\lfloor \frac{p}{2} \right\rfloor m \pmod{p}$$

for some r with $|r| \leq (2\tau + 1/2)(2^\rho - 1) + 1/2$.

Proof. Write $x_i = pq_i + r_i$ with $q_i \in \mathbb{Z}$ and $r_i = [x_i]_p$ for $0 \leq i \leq \tau$. We have $\lfloor \frac{x_1}{2} \rfloor = \frac{pq_1}{2} + \frac{r_1}{2} + \delta$ for $|\delta| \leq 1/2$. Since q_1 is odd, we have, modulo p :

$$c = \sum_{i \in S} x_i + \left\lfloor \frac{x_1}{2} \right\rfloor m - kx_0 = \sum_{i \in S} r_i - kr_0 + \left\lfloor \frac{p}{2} \right\rfloor m + \left(\frac{r_1}{2} + \delta \right) m,$$

for some $k \in [0, \tau]$. Therefore, we have $c = r + \left\lfloor \frac{p}{2} \right\rfloor m \pmod{p}$ for some r with $|r| \leq (2\tau + 1/2)(2^\rho - 1) + 1/2$. \square

Lemma 5 (Addition noise). Let $(sk = p, pk = (x_0, \dots, x_\tau)) \leftarrow \mathbf{AHE.KeyGen}(\lambda)$ and $c_i \leftarrow \mathbf{AHE.Enc}_{pk}(m_i)$ with $c_i = r_i + \lfloor \frac{p}{2} \rfloor m_i \bmod p$ for all $i \in \{1, 2\}$. If $c_{add} \leftarrow \mathbf{AHE.Add}_{x_0}(c_1, c_2)$, then

$$c_{add} = r + \left\lfloor \frac{p}{2} \right\rfloor [m_1 + m_2]_2 \bmod p,$$

for some r with $|r| \leq |r_1 + r_2| + 2^\rho$.

Proof. We have, modulo p :

$$c_{add} = c_1 + c_2 - \delta x_0 = r_1 + r_2 - \delta r_0 + \left\lfloor \frac{p}{2} \right\rfloor [m_1 + m_2]_2 - \delta'$$

for some $\delta, \delta' \in [-1, 1]$. Hence we can write $c_{add} = r + \lfloor \frac{p}{2} \rfloor [m_1 + m_2]_2 \bmod p$ for some r with $|r| \leq |r_1 + r_2| + 2^\rho$. \square

Lemma 6 (Decryption noise). Let p a positive integer and $m \in \{0, 1\}$. Given an integer c , we have

$$\mathbf{AHE.Dec}_p(c) = m \quad \text{if } c = r + \left\lfloor \frac{p}{2} \right\rfloor m \bmod p \quad \text{with } |r| < \frac{p}{4} - \frac{1}{2}.$$

Proof. Write $c = pq + r + \lfloor \frac{p}{2} \rfloor m$. Then, for some $b \in \{0, 1\}$:

$$\left\lfloor c \cdot \frac{2}{p} \right\rfloor = \left\lfloor 2q + m + \frac{2r + b}{p} \right\rfloor = 2q + m + \left\lfloor \frac{2r + b}{p} \right\rfloor,$$

which is congruent to m modulo 2 when $|r| < \frac{p}{4} - \frac{1}{2}$. \square

Theorem 2 (Correctness). Let $\ell \geq 1$. Let $(sk = p, pk = (x_0, \dots, x_\tau)) \leftarrow \mathbf{AHE.KeyGen}(\lambda)$ and $c_i \leftarrow \mathbf{AHE.Enc}_{pk}(m_i)$ for $i = 1, \dots, \ell$ and $m_i \in \{0, 1\}$. Let $c = [\sum_{i=1}^{\ell} c_i]_{x_0}$. Then we have

$$\mathbf{AHE.Dec}_p(c) = \left[\sum_{i=1}^{\ell} m_i \right]_2 \quad \text{when } \ell \leq \frac{2^{\eta-\rho}}{6(4\tau+1)}.$$

In particular, a fresh ciphertext (i.e., with $\ell = 1$) decrypts correctly if $\eta - \rho \geq \log(24\tau + 6)$.

Proof. For $1 \leq i \leq \ell$, write $c_i = pq_i + r_i + \lfloor \frac{p}{2} \rfloor m_i$ for some integers q_i, r_i , and m_i with $|r_i| \leq (2\tau + 1/2)(2^\rho - 1) + 1/2$ and $m_i \in \{0, 1\}$ by Lemma 4. Since $|\sum_{i=1}^{\ell} c_i| \leq \ell x_0/2$, there exists a $k \in \mathbb{Z} \cap [0, \ell/2]$ such that, modulo p :

$$c = \sum_{i=1}^{\ell} r_i - kr_0 + \left\lfloor \frac{p}{2} \right\rfloor \left[\sum_{i=1}^{\ell} m_i \right]_2 + \left\lfloor \frac{p}{2} \right\rfloor \left(\sum_{i=1}^{\ell} m_i - \left[\sum_{i=1}^{\ell} m_i \right]_2 \right).$$

So c is correctly decrypted when $r := \sum_{i=1}^{\ell} r_i - kr_0 + \frac{1}{2} \left(\sum_{i=1}^{\ell} m_i - [\sum_{i=1}^{\ell} m_i]_2 \right)$ is small. By applying Lemmas 4 and 5, we have $|r| \leq (3\ell/2)((2\tau + 1/2)(2^\rho - 1) + 1/2) + \ell/2$. It is less than $\frac{p}{4} - \frac{1}{2}$ if $\ell \leq \frac{2^{\eta-\rho}}{6(4\tau+1)}$. The proof may be completed by using Lemma 6. \square

To guarantee correct decryption, it suffices to take $\eta \geq \rho + \log(24\tau + 6)$.

3.3 Security

We first recall the classical Leftover Hash Lemma (LHL) over finite sums modulo an integer as in [20].

Lemma 7. *Sample $x_1, \dots, x_\tau \leftarrow \mathbb{Z}_{x_0}$ independently, sample $s_1, \dots, s_\tau \leftarrow \{0, 1\}$, and set $y = \sum_{i=1}^\tau s_i x_i \bmod x_0$. Then (x_1, \dots, x_τ, y) is within statistical distance $\frac{1}{2}\sqrt{x_0/2^\tau}$ from $U(\mathbb{Z}_{x_0}^{\tau+1})$.*

This LHL is used to show that the **AHE** ciphertext is computationally indistinguishable from a uniform integer modulo x_0 , independently of the encrypted plaintext bit.

Theorem 3 (Security). *Under the assumptions that $\text{AGCD}_{X, \lfloor D_\alpha \rfloor}(\mathcal{D}_\sigma^*)$ is hard and that $\tau \geq \log X + 2\lambda + 2$, the **AHE** scheme described above is IND-CPA secure.*

Proof. The key generation procedure produces independent $x_i \leftarrow A_{X, \lfloor D_\alpha \rfloor}^{\text{AGCD}}(p)$. With probability exponentially close to 1, there exists i such that x_i is not the largest, and $\lfloor x_i/p \rfloor$ is odd.

Now, in the IND-CPA security experiment, we replace the sampling of the x_i 's by $x_i \leftarrow U(\mathbb{Z} \cap [0, X])$, independently, for $i = 0, \dots, \tau$. We still sort them so that x_0 is the largest, and x_1 is such that $\lfloor x_1/p \rfloor$ is odd (we resample if we cannot find such an x_1). The resulting public key distribution is computationally indistinguishable from the genuine public key distribution, under the assumption that $\text{AGCD}_{X, \lfloor D_\alpha \rfloor}(\mathcal{D}_\sigma^*)$ is hard.

With this modified key generation procedure, the distribution of $(x_i)_{2 \leq i \leq \tau}$ is within exponentially small statistical distance from $U((\mathbb{Z} \cap [0, x_0))^{\tau-1})$. Using Lemma 7 and the assumption on τ , the tuple $(x_2, \dots, x_\tau, \sum_{i>1} s_i x_i \bmod x_0)$ is within exponentially small statistical distance from $U((\mathbb{Z} \cap [0, x_0))^\tau)$. As a result, the distribution of the challenge ciphertext in the IND-CPA experiment is within exponentially small statistical distance from $U(\mathbb{Z} \cap [0, x_0])$, independently of the underlying plaintext. In that experiment, the distinguishing advantage of the adversary is exponentially small. \square

4 A Scale-Invariant AGCD-Based FHE

In this section, we first extend the **AHE** scheme into a somewhat homomorphic scheme allowing a certain amount of homomorphic data manipulation, and then use Gentry's bootstrapping technique [22] to obtain a fully homomorphic encryption scheme.

We adapt some notations from [7] to our context. Let n be a positive integer. Given $x \in \mathbb{Z} \cap [0, 2^n)$ and $y \in \mathbb{R}$, define

$$\begin{aligned} \text{BD}_n(x) &= (x_0, x_1, \dots, x_{n-1}) \in \{0, 1\}^n \text{ with } x = \sum_{i=0}^{n-1} x_i 2^i \\ \mathcal{P}_n(y) &= (y, 2y, \dots, 2^{n-1}y) \in \mathbb{R}^n. \end{aligned}$$

Then we can see that

$$\langle \text{BD}_n(x), \mathcal{P}_n(y) \rangle = \sum_{i=1}^{n-1} x_i(2^i y) = xy.$$

We also recall the definition of a tensor product on the vector space \mathbb{R}^n

$$(u_1, \dots, u_n) \otimes (v_1, \dots, v_n) = (u_1 v_1, u_1 v_2, \dots, u_1 v_n, \dots, u_n v_1, \dots, u_n v_n),$$

and its relation with the inner product

$$\langle \mathbf{u} \otimes \mathbf{u}', \mathbf{v} \otimes \mathbf{v}' \rangle = \langle \mathbf{u}, \mathbf{v} \rangle \cdot \langle \mathbf{u}', \mathbf{v}' \rangle.$$

4.1 The Construction

The scheme **SHE** is identical to **AHE**, except concerning the following two procedures. Its security is inherited from that of **AHE**.

SHE.MultKeyGen(sk). Let $p = sk$. For all $k \in \mathbb{Z} \cap [0, 2\gamma - 2]$, sample $q_{i,j}^*, r_{i,j}^*$ as in $A_{X, \lfloor D_\alpha \rfloor}^{\text{AGCD}}(p)$ and publish a vector $\mathbf{y} = (pq_{i,j}^* + r_{i,j}^*)_{0 \leq i,j < \gamma} + \frac{p}{2} ([\mathcal{P}_\gamma(2/p)]_2 \otimes [\mathcal{P}_\gamma(2/p)]_2)$ as a multiplication key.

SHE.Mult _{x_0, \mathbf{y}} (c_1, c_2). Given two ciphertexts c_1, c_2 , output

$$c_{\text{mult}} := [\langle \text{BD}_\gamma(c_1) \otimes \text{BD}_\gamma(c_2), \mathbf{y} \rangle]_{x_0}.$$

The (i, j) component of the γ^2 -dimensional vector \mathbf{y} is a *fake* encryption of $[2^{i+1}/p]_2 \cdot [2^{j+1}/p]_2$, because it is not decrypted into $[2^{i+1}/p]_2 \cdot [2^{j+1}/p]_2$.

4.2 Correctness

We now prove the correctness of the homomorphic multiplication procedure.

Lemma 8. *Let p be a positive integer. If $c = pq + r + \lfloor p/2 \rfloor m \in \mathbb{Z} \cap [0, 2^\gamma - 2]$ with $q, r \in \mathbb{Z}$ and $m \in \{0, 1\}$, then we have*

$$\langle \text{BD}_\gamma(c), [\mathcal{P}_\gamma(2/p)]_2 \rangle = 2a + m + \varepsilon$$

for an integer a with $|a| \leq (\gamma - \eta + 4)/2$ and a real ε with $|\varepsilon| < (2|r| + 1)/p$.

Proof. Let $\lfloor p/2 \rfloor = (p+b)/2$, $b \in \{0, 1\}$. Then, $2c/p = 2q + m + \varepsilon$, which is equal to $m + \varepsilon$ modulo 2 for $\varepsilon = (2r + b)/p$ with $|\varepsilon| \leq (2|r| + 1)/p$.

Since $\text{BD}_\gamma(c)$ is an integer, we have, modulo 2:

$$\langle \text{BD}_\gamma(c), [\mathcal{P}_\gamma(2/p)]_2 \rangle \equiv \langle \text{BD}_\gamma(c), \mathcal{P}_\gamma(2/p) \rangle = 2c/p.$$

So, $\langle \text{BD}_\gamma(c), [\mathcal{P}_\gamma(2/p)]_2 \rangle = 2a + m + \varepsilon$ for some integer a . Using $2/p + 2^2/p + \dots + 2^{\eta-2}/p = 2(2^{\eta-2} - 1)/p < 1$, we have

$$|\langle \text{BD}_\gamma(c), [\mathcal{P}_\gamma(2/p)]_2 \rangle| \leq \sum_{i=0}^{\gamma-1} \left| \left[\frac{2^{i+1}}{p} \right]_2 \right| \leq \gamma - \eta + 3,$$

which implies $|a| \leq (\gamma - \eta + 4)/2$.

□

Lemma 9 (Multiplication noise). Let $(sk = p, pk = (x_0, \dots, x_\tau)) \leftarrow \mathbf{AHE.KeyGen}(\lambda)$ and $\mathbf{y} \leftarrow \mathbf{SHE.MultKeyGen}(sk)$. Given $c_1, c_2 \in \mathbb{Z} \cap (-x_0/2, x_0/2]$ satisfying $c_i = r_i + \lfloor \frac{p}{2} \rfloor m_i \bmod p$ for $i \in \{0, 1\}$, we have

$$[(\text{BD}_\gamma(c_1) \otimes \text{BD}_\gamma(c_2), \mathbf{y})]_{x_0} = pq + r + \left\lfloor \frac{p}{2} \right\rfloor m_1 m_2$$

for some $q, r \in \mathbb{Z}$ with $|r| < (\gamma - \eta + 6)(|r_1| + |r_2|) + \gamma^2 \cdot 2^{\rho+1}$.

Proof. We have

$$\mathbf{y} = (pq_{i,j}^* + r_{i,j}^{**})_{0 \leq i,j < \gamma} + \frac{p}{2} ([\mathcal{P}_\gamma(2/p)]_2 \otimes [\mathcal{P}_\gamma(2/p)]_2),$$

for some $r_{i,j}^{**} \in r_{i,j}^* + [-1/2, 1/2]$ for all i, j . We now use Lemma 8:

$$\begin{aligned} & \langle \text{BD}_\gamma(c_1) \otimes \text{BD}_\gamma(c_2), \mathbf{y} \rangle \\ &= \langle \text{BD}_\gamma(c_1) \otimes \text{BD}_\gamma(c_2), (pq_{i,j}^* + r_{i,j}^{**})_{i,j} \rangle + \frac{p}{2} \langle \text{BD}_\gamma(c_1), [\mathcal{P}_\gamma(2/p)]_2 \rangle \cdot \langle \text{BD}_\gamma(c_2), [\mathcal{P}_\gamma(2/p)]_2 \rangle \\ &= \sum_{(i,j) \in J} (pq_{i,j}^* + r_{i,j}^{**}) + \frac{p}{2} (m_1 + \varepsilon_1 + 2a_1)(m_2 + \varepsilon_2 + 2a_2), \end{aligned}$$

for some index set $J \subseteq [0, \gamma]^2$, and some $a_1, a_2 \in \mathbb{Z}$, $\varepsilon_1, \varepsilon_2 \in \mathbb{R}$ that satisfy $|a_1|, |a_2| \leq (\gamma - \eta + 4)/2$, $|\varepsilon_1| < (2|r_1| + 1)/p$ and $|\varepsilon_2| < (2|r_2| + 1)/p$. Since $\frac{p}{2}((m_1 + 2a_1)(m_2 + 2a_2) - m_1 m_2)$ is a multiple of p , we have that, for some integer q

$$[(\text{BD}_\gamma(c_1) \otimes \text{BD}_\gamma(c_2), \mathbf{y})]_{x_0} = pq + r + \left\lfloor \frac{p}{2} \right\rfloor m_1 m_2,$$

where

$$r = \sum_{(i,j) \in J} r_{i,j}^{**} + \frac{p}{2} (\varepsilon_2(m_1 + 2a_1) + \varepsilon_1(m_2 + 2a_2) + \varepsilon_1 \varepsilon_2) - \frac{1}{2} m_1 m_2 - kr_0$$

for some $k \in [-1, \gamma^2]$. Therefore, we have $|r| < \gamma^2 \cdot 2^{\rho+1} + (\gamma - \eta + 6)(|r_1| + |r_2|)$. Note that r is an integer because all of $\text{BD}_\gamma(c_1)$, $\text{BD}_\gamma(c_2)$ and \mathbf{y} has only integer components. \square

Let $c_i \leftarrow \mathbf{SHE.Enc}_{pk}(m_i)$ with $c_i = r_i + \lfloor \frac{p}{2} \rfloor m_i \bmod p$ for $i \in \{1, 2\}$, $c_{\text{add}} \leftarrow \mathbf{SHE.Add}_{pk}(c_1, c_2)$ and $c_{\text{mult}} \leftarrow \mathbf{SHE.Mult}_{pk}(c_1, c_2)$. From Lemmas 5 and 9, we can see that

$$\begin{aligned} c_{\text{add}} &= r_{\text{add}} + \left\lfloor \frac{p}{2} \right\rfloor [m_1 + m_2]_2 \bmod p \\ c_{\text{mult}} &= r_{\text{mult}} + \left\lfloor \frac{p}{2} \right\rfloor [m_1 m_2]_2 \bmod p, \end{aligned}$$

with $|r_{\text{add}}| \leq |r_1| + |r_2| + 2^\rho$ and $|r_{\text{mult}}| \leq (\gamma - \eta + 6)(|r_1| + |r_2|) + \gamma^2 \cdot 2^{\rho+1}$. Both the addition and multiplication in our scheme increase noise only *additively*.

Definition 5. A scheme **HE** is L -homomorphic if for any depth L binary circuit \mathcal{C} and any set of inputs $m_1, \dots, m_\ell \in \{0, 1\}$, it holds that

$$\mathbf{HE}.\mathbf{Dec}_{sk}(\mathbf{HE}.\mathbf{Eval}_{evk}(\mathcal{C}, (c_1, \dots, c_\ell))) = \mathcal{C}(m_1, \dots, m_\ell)$$

with probability $\geq 1 - \lambda^{-\omega(1)}$, where $(pk, evk, sk) \leftarrow \mathbf{HE}.\mathbf{KeyGen}(\lambda)$ and $c_i \leftarrow \mathbf{HE}.\mathbf{Enc}_{pk}(m_i)$ for all $i \leq \ell$.

Theorem 4. The scheme **SHE** is L -homomorphic if

$$\eta - \rho \geq L(1 + \log(\gamma - \eta + 6)) + 3 + \log\left(2\tau + \frac{\gamma^2}{\gamma - \eta + 6}\right).$$

Proof. For each $i \in [0, L]$, let c_i be a ciphertext with $c_i = r_i + \lfloor \frac{p}{2} \rfloor m_i \bmod p$ after the evaluation of the i -th level gates. Let R_i be a bound on the noise magnitude $|r_i|$. First, we have $R_0 = (2\tau + 1/2)(2^\rho - 1) + 1/2$ by Lemma 4. By Lemmas 5 and 9, we have that $R_{i+1} := 2(\gamma - \eta + 6)R_i + \gamma^2 \cdot 2^{\rho+1}$ is a valid level $(i+1)$ bound (for all $i \geq 0$). By solving the recurrence equation, we obtain

$$R_L \leq \left(2\tau + \frac{\gamma^2}{\gamma - \eta + 6}\right) 2^{\rho+1} 2^L (\gamma - \eta + 6)^L - \frac{\gamma^2 \cdot 2^{\rho+1}}{\gamma - \eta + 6},$$

which is at most $\frac{p}{4} - \frac{1}{2}$ if $(\eta - \rho)$ satisfies the condition. In that case, any ciphertext after evaluation of any circuit of depth L can be correctly decrypted. \square

Combining with $\rho = \lambda$, $\gamma = \Omega(\frac{\lambda}{\log \lambda}(\eta - \rho)^2)$, $\tau = \gamma + \Omega(\lambda)$, we may take $\gamma = \Theta(\lambda L^2 \log \lambda)$. Note that the ciphertext size γ is quasi-linear in the security parameter λ .

4.3 Bootstrapping

We provide a bound on the multiplicative depth of the decryption circuit corresponding to $\mathbf{AHE}.\mathbf{Dec}_p(c) = [[2c/p]]_2$.

We take an approximation z to $2/p$ such that $|z - \frac{2}{p}| < 2^{-(\gamma+\eta)}$. Write $z = \sum_{i=0}^{\gamma+\eta} z_{-i} 2^{-i}$ for $z_{-i} \in \{0, 1\}$ for each i . As $c \in [0, 2^\gamma - 2]$, we have $|cz - 2c/p| < 2^{-\eta}$. Therefore, we have $[[cz]]_2 = m$ when $c = pq + r + \lfloor \frac{p}{2} \rfloor m$ and $|r| < \frac{p}{4} - \frac{1}{2}$. Since the η most significant bits of z are zero, the most expensive step in decryption consists in adding up to γ integers of bit-lengths $\leq 2\gamma$. This can be implemented with a binary circuit of $O(\log \gamma)$ depth.

By Theorem 4 and [22], **SHE** is bootstrappable and may be turned into a fully homomorphic encryption scheme when $\eta - \rho = \Omega(\log^2 \gamma)$. For bootstrapping we publish encryptions of z_i 's as a bootstrapping key. This requires space $O(\gamma^2)$.

5 Truncation of Ciphertexts

Since the message bit is embedded into the most significant bit of the ciphertext modulo p , some least significant bits of the ciphertext are irrelevant to

decryption correctness. However as we truncate more least significant bits of a ciphertext, decryption failure probability increases slightly at first and becomes overwhelming after some point between ρ and η bits of truncation.

Let $N = 2^\nu$ for a positive integer $\nu < \gamma$. Given a ciphertext c , define $\hat{c} = \lfloor c/N \rfloor$ so that $|N\hat{c} - c| \leq N/2$. In the following, the quantity \hat{c} can play a similar role to that of the corresponding ciphertext c in each component of **SHE**, for appropriate ν .

1. In the encryption stage, given $\hat{c} := [\sum_{i \in S} \hat{x}_i + \lfloor \frac{\hat{x}_1}{2} \rfloor m]_{\hat{x}_0}$ for some $S \subseteq \mathbb{Z} \cap [1, \tau]$, we have, for some integer k :

$$N\hat{c} = \sum_{i \in S} x_i + \lfloor \frac{x_1}{2} \rfloor m - kx_0 + N \sum_{i \in S} (\hat{x}_i - x_i/N) + Nm \left(\lfloor \frac{\hat{x}_1}{2} \rfloor - \lfloor \frac{x_1}{2} \rfloor / N \right) - kN(\hat{x}_0 - x_0/N),$$

which is equivalent to $r + \lfloor \frac{p}{2} \rfloor m$ modulo p for $|r| \leq (2^{\rho+1} + N)(\tau + 1)$.

2. In the decryption stage, given $\hat{c} = \lfloor c/N \rfloor$ we have

$$\left[\left\lfloor \frac{2\hat{c}}{p/N} \right\rfloor \right]_2 = m$$

if $|r| < (p - N)/4 - 1/2$ when $c = r + \lfloor \frac{p}{2} \rfloor m$ mod p .

3. In the addition stage, given \hat{c}_1 and \hat{c}_2 , we define $\hat{c}_{\text{add}} = [\hat{c}_1 + \hat{c}_2]_{\hat{x}_0}$. Then we can show (similarly to Lemma 5):

$$N\hat{c}_{\text{add}} = r + \left\lfloor \frac{p}{2} \right\rfloor [m_1 + m_2]_2 \text{ mod } p$$

for $|r| \leq |r_1| + |r_2| + (2^\rho + \frac{3}{2}N)$ when $c_i = pq_i + r_i + \lfloor \frac{p}{2} \rfloor m$ for $i \in \{1, 2\}$.

4. In the multiplication stage, we use $\hat{\mathbf{y}}$ to be the vector of bit-length $(\gamma - \nu)^2$ obtained by removing all entries (i, j) of \mathbf{y} such that $i < \nu$ or $j \leq \nu$. Given \hat{c}_1 and \hat{c}_2 , we set

$$\hat{c}_{\text{mult}} = [\text{BD}_{\hat{\gamma}}(\hat{c}_1) \otimes \text{BD}_{\hat{\gamma}}(\hat{c}_2), \hat{\mathbf{y}}]_{\hat{x}_0},$$

where $\hat{\gamma} = \gamma - \nu$. We can show that for all $i \in \{1, 2\}$, we have $\langle \text{BD}_{\hat{\gamma}}(\hat{c}_i), [\mathcal{P}_{\hat{\gamma}}(2N/p)]_2 \rangle = \frac{2\hat{c}_i}{p/N} = 2a_i + m_i + \varepsilon_i$ for $a_i \in \mathbb{Z}$ with $|a_i| \leq (\hat{\gamma} - \eta + 4)/2$, $m_i \in \{0, 1\}$ and $\varepsilon_i \in \mathbb{R}$ with $|\varepsilon_i| < \frac{2(|r|+N)}{p}$. Thus $\hat{c}_{\text{mult}} = pq + r + \lfloor \frac{p}{2} \rfloor m_1 m_2$ for some $q, r \in \mathbb{Z}$ satisfying

$$|r| < (\hat{\gamma} - \eta + 6)(|r_1| + |r_2|) + \hat{\gamma}^2 \cdot 2^{\rho+1},$$

when $c_i = r_i + \lfloor \frac{p}{2} \rfloor m_i$ mod p for $i \in \{1, 2\}$.

5. In the bootstrapping stage, we take $z \in 2^{-(\gamma+\eta-\rho)}$ to be an approximation of $2/p$ with $|z - \frac{2}{p}| < 2^{-(\gamma+\eta-\rho)}$. We have

$$\left\| N\hat{c}z - \frac{2}{p}c \right\| \leq \left(c + \frac{1}{N} \right) \left(\frac{2}{p} + 2^{-(\gamma+\eta-\rho)} \right) \leq \frac{(2^{\rho+1} + N)}{p}.$$

Combining with $\frac{2c}{p} = 2q + m + \frac{2r+m}{p}$ for $c = pq + r + \lfloor \frac{p}{2} \rfloor m$, we have $\lfloor [N\hat{c}z] \rfloor_2 = m$ if $(2r + m + 2^{\rho+1} + N)/p < 1/2$. It is satisfied when $|r| < p/4 - 2^\rho - (N+1)/2$. If $N \leq p/4$, we have a similar homomorphic capacity of Theorem 4 and so **SHE** becomes bootstrappable similarly. In this case, however, the decryption can be done with a binary circuit of $O(\log(\gamma - \rho))$ depth.

In the above observations, we can see that encryption noise and addition noise are almost the same when $\nu \leq \rho$ and the decryption and the bootstrapping work similarly when $\nu < \eta - 1$. For multiplication, as ν grows, the multiplication error decreases. Hence truncating ciphertexts by ρ bits results in similar performance, but with reduced ciphertext bit-length. In that setup, the bit-size of ciphertext becomes $\gamma - \rho$.

The known attacks on AGCD do not say much on the complexity of AGCD when $\gamma - \rho$ is small. A naive attack is as follows. Given $c = pq + r$, we first guess the $\gamma - \eta$ bits of q and then compute $\lfloor \frac{c}{q} \rfloor = p + \lfloor \frac{r}{q} \rfloor$. Since $\frac{r}{q} < 2^{\rho-(\gamma-\eta)}$, we can obtain the $\gamma - \rho$ most significant bits of p . This is significant. To avoid this attack, we need to set $\gamma - \eta \geq \lambda$. In that case, the ciphertext size is $\approx \gamma - \rho = (\gamma - \eta) + (\eta - \rho) \geq \lambda + \Omega(L \log \lambda)$.

Note that this truncation method is different from decreasing the bit-size ρ of the noise. If ρ is set smaller, then the ciphertext bit-length γ should be increased to resist lattice-based attacks, i.e., it must satisfy $\gamma \geq \Omega(\frac{\lambda}{\log \lambda}(\eta - \rho)^2)$. If we reduce ρ and η simultaneously, resistance against the lattice-based attacks can be maintained, but the scheme becomes susceptible to exhaustive search on the noise components r_i .

Acknowledgments. The authors thank Miran Kim, Adeline Langlois, Yongsoo Song and Ron Steinfeld for helpful discussions. We thank Oded Regev for his sketch of the SILWE to ZDLWE reduction [43]. The first author was supported by the ICT R&D program of MSIP/IITP [No. 10047212]. The second author was supported by the ERC Starting Grant ERC-2013-StG-335086-LATTAC.

A Some Useful Lemmas on Lattice Gaussians

The first statement of the following lemma is a special case of [33, Le. 4.1]. The second statement can be obtained by a simple calculation exploiting [40, Claim 3.8].

Lemma 10. *Let $r, \varepsilon > 0$ such that $r \geq \Omega(\sqrt{\ln(1/\varepsilon)})$. Then the distribution $D_r \bmod 1$ is within statistical distance $O(\varepsilon)$ from $U(\mathbb{T})$. If $x \leftarrow D_r$, then the distribution of x conditioned on $x \bmod 1$ is within statistical distance $O(\varepsilon)$ of $D_{\mathbb{Z},r}$.*

Lemma 11 (Special case of [26, Cor. 2.8]). *Let $q \geq 1$ and $r, \varepsilon > 0$ such that $r \geq \Omega(\sqrt{\ln(1/\varepsilon)})$. Then the distribution $D_{\mathbb{Z}/q,r} \bmod 1$ is within statistical distance $O(\varepsilon)$ from $U(\mathbb{T}_q)$.*

Lemma 12 (Special case of [40, Cor. 3.10]). *Let $n \geq 1, \mathbf{z} \in \mathbb{R}^n$ and $r, \varepsilon, \alpha > 0$ with $(1/r^2 + \|\mathbf{z}\|^2/\alpha^2)^{-1/2} \geq \Omega(\sqrt{\ln(n/\varepsilon)})$. Then the distribution of $\langle D_{\mathbb{Z},r}^n, \mathbf{z} \rangle + D_\alpha$ is within statistical distance $O(\varepsilon)$ from D_β with $\beta = (\alpha^2 + \|\mathbf{z}\|^2 r^2)^{1/2}$.*

Lemma 13 (Special case of [37, Th. 3.1]). *Let $r, q, s, \varepsilon > 0$ such that $r \geq \Omega(\sqrt{\ln(1/\varepsilon)}/q)$. Sample $x \leftarrow D_s$ and $k \leftarrow D_{\mathbb{Z}/q,r,x}$. Then the distribution of k is within statistical distance $O(\varepsilon)$ from $D_{\mathbb{Z}/q,(r^2+s^2)^{1/2}}$ and, conditioned on k , the distribution of x is within statistical distance $O(\varepsilon)$ from $k \frac{1}{(1+r^2/s^2)^{1/2}} + D_{(r^{-2}+s^{-2})^{-1/2}}$.*

We also use the following result, which can be derived from Lemmas 10, 11 and 13.

Lemma 14. *Let $r, q, \varepsilon > 0$ such that $r \geq \Omega(\sqrt{\ln(1/\varepsilon)}/q)$. Sample $x \leftarrow \mathbb{T}$ and $k \leftarrow D_{\mathbb{Z}/q,r,x}$. Then $k \bmod 1$ is uniformly distributed over \mathbb{T}_q and the distribution of x conditioned on k is within statistical distance $O(\varepsilon)$ from $k + D_r \bmod 1$.*

Proof. Let $s \geq \Omega(\sqrt{\ln(1/\varepsilon)}/q)$. By Lemma 10, the uniform distribution over \mathbb{T} is within statistical distance $O(\varepsilon)$ from the distribution $D_s \bmod 1$. By Lemma 13, the distribution of k is within statistical distance $O(\varepsilon)$ from $D_{\mathbb{Z}/q,(r^2+s^2)^{1/2}} \bmod 1$ and conditioned on k the distribution of x is within statistical distance $O(\varepsilon)$ from $k \frac{1}{(1+r^2/s^2)^{1/2}} + D_{(r^{-2}+s^{-2})^{-1/2}} \bmod 1$. The proof can be completed by using Lemma 11 and letting s tend to infinity. \square

B Converse Results for Lemmas 1, 2 and 3

Lemma 15. *Let $\alpha, \beta \in (0, 1)$, $m, n, q, B \geq 1$ and \mathcal{D} a distribution over \mathbb{Z}^n . Assume that there exists an $(\varepsilon_1, \varepsilon_2)$ -distinguisher for $\text{LWE}_{n,q,D_{\leq \beta}}^m(\mathcal{D})$. If \mathcal{D} is $(B, \varepsilon_2/2)$ -bounded, and $\alpha \leq \beta - \Omega(\sqrt{\ln(mn/\varepsilon_1)}B/q)$, then there exists an $(\Omega(\varepsilon_1), \Omega(\varepsilon_2))$ -distinguisher for $\text{SILWE}_{n,D_{\leq \alpha}}^m(\mathcal{D})$.*

Proof. The reduction architecture is similar to the one of Lemma 1. We map each input sample (\mathbf{a}, b) for $\text{SILWE}_{n,D_{\leq \alpha}}$ to an input samples (\mathbf{a}', b') for $\text{LWE}_{n,q,D_{\leq \beta}}$ as follows: Sample $\mathbf{a}' \leftarrow D_{\mathbb{Z}^n/q,r,\mathbf{a}}$ with $r = \Omega(\sqrt{\ln(mn/\varepsilon_1)})/q$ (for this, we independently sample each coordinate $a'_j \leftarrow D_{\mathbb{Z}/q,r,a_j}$); set $b' = b$.

By Lemma 14, we have that the distribution of \mathbf{a}' is within statistical distance $O(\varepsilon_1/m)$ from $U(\mathbb{T}_q^n)$, and that conditioned on \mathbf{a}' , the distribution of $\mathbf{f} := \mathbf{a} - \mathbf{a}'$ is within statistical distance $O(\varepsilon_1/m)$ from D_r . As a result, the transformation maps the uniform distribution over $\mathbb{T}^n \times \mathbb{T}$ to a distribution within statistical distance $O(\varepsilon_1/m)$ from the uniform distribution over $\mathbb{T}_q^n \times \mathbb{T}$. Further, if $b = \langle \mathbf{a}, \mathbf{s} \rangle + e$ for some fixed \mathbf{s} and $e \leftarrow D_{\leq \alpha}$, then $b' = b = \langle \mathbf{a}', \mathbf{s} \rangle + \langle \mathbf{f}, \mathbf{s} \rangle + e$. Conditioned on \mathbf{a}' , the distribution of $\langle \mathbf{f}, \mathbf{s} \rangle + e$ is within statistical distance $O(\varepsilon_1/m)$ of $D_{\leq \sqrt{\alpha^2 + \|\mathbf{s}\|^2} \cdot \sqrt{2}}$. We have $\|\mathbf{s}\| \leq B$ with probability $\geq 1 - \varepsilon_2/2$ over the randomness of $s \leftarrow \mathcal{D}$. This allows to complete the proof. \square

Lemma 16 (Adapted from [42, App. A]). *Let $\alpha, \beta \in (0, 1)$, $B \geq 1$ and \mathcal{D} a distribution over \mathbb{Z} . Assume that there exists an $(\varepsilon_1, \varepsilon_2)$ -distinguisher for $\text{SILWE}_{1, D_{\leq \beta}}^m(\mathcal{D})$. If \mathcal{D} is $(B, \delta, \varepsilon_2/2)$ -contained and $\alpha \leq O(\beta)$, then there exists an $(\Omega(\varepsilon_1), \Omega(\varepsilon_2 \delta \alpha / \sqrt{\ln(m/\varepsilon_1)}))$ -distinguisher for $\text{ZDLWE}_{D_{\leq \alpha}}^m(\mathcal{D})$.*

Proof. We reduce ZDLWE to SILWE. Let $r = \Theta(\sqrt{\ln(m/\varepsilon_1)})$ (chosen to be able to use Lemma 10) and $\delta' = \Theta(\delta \alpha / \sqrt{\ln(m/\varepsilon_1)})$. The reduction produces a guess s' of the ZDLWE secret s by sampling $s' \leftarrow B\delta' \cdot (\mathbb{Z} \cap [0, \lceil 1/\delta' \rceil])$; then it maps any input sample y for $\text{ZDLWE}_{D_{\leq \alpha}}^m$ to an input sample (a, b) for $\text{SILWE}_{1, D_{\leq \beta}}^m$, as follows: Sample $f \leftarrow D_r$; set $a = [y + f/s']_1$ and $b = [f]_1$.

We assume that $|s| \in [\delta B, B]$, $|s' - s| \leq B\delta'$ and that the $\text{SILWE}_{1, D_{\leq \beta}}^m$ distinguisher succeeds. As in the proof of Lemma 2, this event has weight $\Omega(\varepsilon_2 \delta')$.

Assume that y is uniformly distributed in \mathbb{T} . By Lemma 10, the distribution of b is within statistical distance $O(\varepsilon_1/m)$ from uniform, independently of y . Therefore, the distribution of the pair (a, b) is within statistical distance $O(\varepsilon_1/m)$ from uniform.

Now, assume that $y = (k + e)/s$ with $k \leftarrow \mathbb{Z} \cap [0, s)$ and $e \leftarrow D_{\leq \alpha}$. We have $a = (k + e)/s + f/s'$ and $b - as = [-e + f(1 - s/s')]_1$. Let $f' = fs/s'$. By Lemma 10, the distribution of $a' := (k + f')/s$ is within statistical distance $O(\varepsilon_1/m)$ from uniform and the distribution of f' conditioned on a' is within statistical distance $O(\varepsilon_1/m)$ of $D_{\mathbb{Z}, r|s/s'|}$. The assumption of Lemma 10 holds because $r|s/s'| \geq \Omega(r) \geq \Omega(\sqrt{\ln(m/\varepsilon_1)})$, thanks to the choices of δ' and r . By Lemma 12, the distribution of $b - as = -e + f'(s'/s - 1)$ is within statistical distance $O(\varepsilon_1/m)$ of $D_{\leq \beta'}$ with $\beta' = \sqrt{\alpha^2 + r^2(1 - s/s')^2}$, assuming that $((s'/rs)^2 + (s'/s - 1)^2/\alpha^2)^{-1/2} \geq \Omega(\sqrt{\ln(m/\varepsilon_1)})$. As $|s/s'| \geq \Omega(1)$ and $|s'/s - 1| \leq O(\delta'/\delta)$, the definitions of r and δ' imply that the latter condition holds. Further, as $|1 - s/s'| \leq O(\delta'/\delta)$, we have that $\beta' \leq O(\alpha + r\delta'/\delta)$. This completes the proof. \square

Lemma 17. *Let $\alpha, \beta \in (0, 1)$, $X, B \geq 1$ and \mathcal{D} a distribution over \mathbb{Z} . Assume that there exists an $(\varepsilon_1, \varepsilon_2)$ -distinguisher for $\text{ZDLWE}_{D_{\leq \beta}}^m(\lfloor X/\mathcal{D} \rfloor)$. If \mathcal{D} is $(B, \delta, \varepsilon_2/2)$ -contained, $X \geq \Omega(mB/\varepsilon_1)$, $\alpha \geq \Omega(Bm/(\delta\varepsilon_1))$ and $\beta \geq \alpha B/X$, then there exists an $(\Omega(\varepsilon_1), \Omega(\varepsilon_2))$ -distinguisher for $\text{AGCD}_{X, \lfloor D_{\leq \alpha} \rfloor}^m(\mathcal{D})$.*

Proof. Given an input sample x for $\text{AGCD}_{X, \lfloor D_{\alpha} \rfloor}$, the reduction produces an input sample y for $\text{ZDLWE}_{D_{\leq \alpha}}^m$ as follows: Sample $f \leftarrow \mathbb{T}$; set $y = (x + f)/X$. If x is uniformly distributed over $\mathbb{Z} \cap [0, X)$, then so is y over \mathbb{T} .

Now, assume that $x = qp + r$ for some fixed $p = \lfloor X/s \rfloor$ (with s sampled from \mathcal{D}), $q \leftarrow \mathbb{Z} \cap [0, X/p)$ and $r \leftarrow \lfloor D_{\leq \alpha} \rfloor$. We have $y = (q + e + \Delta)/s$ with $s = \lfloor X/p \rfloor$, $e = (r + f)s/X$ and $\Delta = \varepsilon qs/X$ for some $\varepsilon \in [-1/2, 1/2]$.

As $\alpha \geq \Omega(\sqrt{\ln(m/\varepsilon_1)})$, the distribution of e is within statistical distance $O(\varepsilon_1/m)$ from $D_{\leq \alpha|s|/X}$. Further, we have $|\Delta| \leq |s|/p \leq O(B^2/X)$ (assuming that $|s| \leq B$). Thanks to the assumptions on \mathcal{D} and α , we have $|\Delta| \leq O(\frac{\varepsilon_1}{m} \frac{\alpha|s|}{X})$, and the term $e + \Delta$ is within statistical distance $O(\varepsilon_1/m)$ from $D_{\leq \alpha|s|/X}$. Note that $\alpha B/X \leq \beta$.

It now suffices to show that the distributions $U(\mathbb{Z} \cap [0, s))$ and $U(\mathbb{Z} \cap [0, X/p))$ are within statistical distance $O(\varepsilon_1/m)$. The proof is identical to that of Lemma 3. \square

C Orthogonal Lattice Attack on AGCD

Let us recall the orthogonal lattice attack on AGCD in [20].

Suppose we are given samples $(x_i = pq_i + r_i)_{1 \leq i \leq m}$ from $A_{X, \lfloor D_\alpha \rfloor}^{\text{AGCD}}(p)$. Let 2^ρ be an upper bound on the magnitudes of the r_i 's. Consider the integral lattice \mathcal{L} generated by the rows of the following $m \times (m+1)$ matrix:

$$\begin{bmatrix} x_1 & 2^\rho \\ x_2 & 2^\rho \\ \vdots & \ddots \\ x_m & 2^\rho \end{bmatrix}$$

Define the vector $\mathbf{u} := (1, -\frac{r_1}{2^\rho}, \dots, -\frac{r_d}{2^\rho})$. For any element $\mathbf{v} \in \mathcal{L}$, we have $\langle \mathbf{u}, \mathbf{v} \rangle \equiv 0 \pmod{p}$. Further, if $\|\mathbf{v}\|_1 < p$, then we have $|\langle \mathbf{u}, \mathbf{v} \rangle| \leq \|\mathbf{v}\|_1 < p$, since each component of \mathbf{u} is at most 1. That is, we have $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ over \mathbb{Z} . Hence if we find m linearly independent vectors \mathbf{v} in \mathcal{L} with $\|\mathbf{v}\|_1 < p$, we can recover \mathbf{u} and hence find p from $\gcd(x_1 - r_1, \dots, x_m - r_m)$ with overwhelming probability.

The lattice \mathcal{L} has determinant $\approx 2^{\gamma+(m-1)\rho}$. Assuming that all minima are almost equal, their norms are $\approx 2^{\gamma/m+\rho(m-1)/m}$. In time 2^λ , lattice reduction [2, 44] allows to find m linearly independent lattice vectors of norms $\approx \lambda^{O(m/\lambda)} \cdot 2^{\rho(m-1)/m+\gamma/m}$. The optimal choice for m is $\approx \Theta(\sqrt{\gamma\lambda}/\log\lambda)$, which leads to vector norms that are $\approx 2^{O(\sqrt{\gamma\log\lambda/\lambda})+\rho}$. The attack is thwarted if $\gamma \geq \Omega(\frac{\lambda}{\log\lambda})(\eta - \rho)^2$.

The Simultaneous Diophantine Approximation algorithm for AGCD (see [20]) has similar performance.

References

1. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: Proc. of STOC, pp. 284–293. ACM (1997)
2. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proc. of STOC, pp. 601–610. ACM (2001)
3. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 297–314. Springer, Heidelberg (2014)
4. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009)
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012)

6. Brakerski, Z., Gentry, C., Halevi, S.: Packed ciphertexts in LWE-based homomorphic encryption. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 1–13. Springer, Heidelberg (2013)
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Proc. of ITCS, pp. 309–325. ACM (2012)
8. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Proc. of STOC, pp. 575–584. ACM (2013)
9. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In Proc. of FOCS, pp. 97–106. IEEE Computer Society Press (2011)
10. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011)
11. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. **43**(2), 831–871 (2014)
12. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Proc. of ITCS, pp. 1–12. ACM (2014)
13. Chen, Y., Nguyen, P.Q.: Faster algorithms for approximate common divisors: breaking fully-homomorphic-encryption challenges over the integers. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 502–519. Springer, Heidelberg (2012)
14. Cheon, J.H., Coron, J.-S., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch fully homomorphic encryption over the integers. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 315–335. Springer, Heidelberg (2013)
15. Cohn, H., Heninger, N.: Approximate common divisors via lattices. Cryptology ePrint Archive, Report 2011/437 (2011). <http://eprint.iacr.org/>
16. Coron, J.-S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 476–493. Springer, Heidelberg (2013)
17. Coron, J.-S., Lepoint, T., Tibouchi, M.: Scale-invariant fully homomorphic encryption over the integers. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 311–328. Springer, Heidelberg (2014)
18. Coron, J.-S., Mandal, A., Naccache, D., Tibouchi, M.: Fully homomorphic encryption over the integers with shorter public keys. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 487–504. Springer, Heidelberg (2011)
19. Coron, J.-S., Naccache, D., Tibouchi, M.: Public key compression and modulus switching for fully homomorphic encryption over the integers. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 446–464. Springer, Heidelberg (2012)
20. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
21. Gentry, C.: A fully homomorphic encryption scheme. PhD thesis, Stanford University(2009). <http://crypto.stanford.edu/craig>
22. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proc. of STOC, pp. 169–178. ACM (2009)

23. Gentry, C., Halevi, S.: Implementing Gentry's fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011)
24. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (2012)
25. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012)
26. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proc. of STOC, pp. 197–206. ACM (2008). Full version available at <http://eprint.iacr.org/2007/432.pdf>
27. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013)
28. Howgrave-Graham, N.: Approximate integer common divisors. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 51–66. Springer, Heidelberg (2001)
29. Kim, J., Lee, M.S., Yun, A., Cheon, J.H.: CRT-based fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2013/057 (2013). <http://eprint.iacr.org/>
30. Lagarias, J.C.: The computational complexity of simultaneous diophantine approximation problems. SIAM J. Comput. **14**(1), 196–209 (1985)
31. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM **60**(6), 43 (2013)
32. 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)
33. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput. **37**(1), 267–302 (2007)
34. Micciancio, D., Regev, O.: Lattice-based cryptography. In: Bernstein, D. J., Buchmann, J., Dahmen, E. (eds) Post-Quantum Cryptography, pp. 147–191. Springer (2009)
35. Nguyen, P.Q., Stern, J.: The two faces of lattices in cryptology. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 146–180. Springer, Heidelberg (2001)
36. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: Proc. of STOC, pp. 333–342. ACM (2009)
37. 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)
38. Regev, O.: New lattice based cryptographic constructions. In: Proc. of STOC, pp. 407–416. ACM (2003)
39. Regev, O.: New lattice-based cryptographic constructions. J. ACM **51**(6), 899–942 (2004)
40. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proc. of STOC, pp. 84–93. ACM (2005)
41. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6) (2009)
42. Regev, O.: The learning with errors problem. Invited survey in CCC 2010 (2010). <http://www.cims.nyu.edu/regev/>

43. Regev, O.: Private communication (2012)
44. Schnorr, C.P.: A hierarchy of polynomial lattice basis reduction algorithms. *Theor. Comput. Science* **53**, 201–224 (1987)
45. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010)
46. Stehlé, D., Steinfeld, R.: Faster fully homomorphic encryption. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 377–394. Springer, Heidelberg (2010)

(Batch) Fully Homomorphic Encryption over Integers for Non-Binary Message Spaces

Koji Nuida^{1,2(✉)} and Kaoru Kurosawa³

¹ National Institute of Advanced Industrial Science and Technology (AIST),
Tsukuba, Ibaraki 305-8568, Japan

² Japan Science and Technology Agency (JST) PRESTO Researcher, Nagoya, Japan
`k.nuida@aist.go.jp`

³ Ibaraki University, Hitachi, Ibaraki 316-8511, Japan
`kurosawa@mx.ibaraki.ac.jp`

Abstract. In this paper, we construct a fully homomorphic encryption (FHE) scheme over integers with the message space \mathbb{Z}_Q for any prime Q . Even for the binary case $Q = 2$, our decryption circuit has a smaller degree than that of the previous scheme; the multiplicative degree is reduced from $O(\lambda(\log \lambda)^2)$ to $O(\lambda)$, where λ is the security parameter. We also extend our FHE scheme to a batch FHE scheme.

Keywords: Fully homomorphic encryption · Non-binary message

1 Introduction

Fully homomorphic encryption (FHE) enables computation of any function on the encrypted data. Many FHE schemes appeared recently after the first construction of Gentry [9]. In [9], the following general framework for constructing FHE schemes was also presented. (1) Construct a somewhat homomorphic encryption (SHE) scheme which can evaluate a limited class of functions homomorphically. (2) Transform (or *squash*) the SHE scheme into a bootstrappable scheme whose decryption circuit has a low enough multiplicative degree. (3) Apply Gentry’s transformation to get an FHE scheme from the bootstrappable scheme.

At Eurocrypt 2010, van Dijk et al. [8] constructed an “FHE scheme over the integers”. At Eurocrypt 2013, Cheon et al. [3] extended it to a batch FHE scheme, where the message space is extended from \mathbb{Z}_2 to $(\mathbb{Z}_2)^k$. In [3], they also presented a batch *SHE* scheme for the message space $\mathbb{Z}_{Q_1} \times \cdots \times \mathbb{Z}_{Q_k}$. However, FHE has not been achieved for the case of primes $Q_i > 2$, even for the non-batch case $k = 1$.

1.1 What Is the Problem?

Let λ be the security parameter, and let \mathcal{M} denote the message space. In the scheme of van Dijk et al. [8], $\mathcal{M} = \mathbb{Z}_2$ and the ciphertext c of a plaintext $m \in \mathcal{M}$ is $c = pq + 2r + m$, where p is a secret prime and r is a small noise. In their SHE scheme, the decryption is given by $m = (c \bmod p) \bmod 2 = c - p$.

$[c/p] \bmod 2 = c - [c/p] \bmod 2$. In the bootstrappable scheme, the (squashed) decryption algorithm works as

$$m \leftarrow (c \bmod 2) \oplus \left(\left[\sum_{i=1}^{\Theta} s_i z_i \right] \bmod 2 \right) . \quad (1)$$

Here $(s_1, \dots, s_\Theta) \in \{0, 1\}^\Theta$ is the secret key with Hamming weight λ and each $z_i = (z_{i,0} z_{i,1} \dots z_{i,L})_2$ is a real number with $L = \lceil \log_2 \lambda \rceil + 3$ bits of precision after the binary point, satisfying $\sum_{i=1}^{\Theta} s_i z_i \approx c/p$.¹ They constructed a low multiplicative degree circuit computing $(\lfloor \sum_{i=1}^{\Theta} s_i z_i \rfloor \bmod 2)$ in (1) by two steps [8]:

1. The first circuit computes $W_j = \sum_{i=1}^{\Theta} s_i z_{i,j}$ for $j = 0, 1, \dots, L$. Hence

$$\sum_{i=1}^{\Theta} s_i z_i = W_0 + 2^{-1}W_1 + \dots + 2^{-L}W_L .$$

2. By applying the three-for-two trick repeatedly, the second circuit computes a and b satisfying

$$W_0 + 2^{-1}W_1 + \dots + 2^{-L}W_L = a + b \bmod 2 .$$

The multiplicative degree of the first circuit is λ since $W_j \leq \lambda$, and it is $O((\log \lambda)^2)$ for the second circuit. Hence, the total degree of the decryption circuit is $O(\lambda(\log \lambda)^2)$; see also the second last paragraph of the proof of Theorem 6.2 in [8] for the evaluation of the total degree.

Now to compute W_j , we have to homomorphically compute (at least) a half adder; it needs a pair of polynomials, one for the sum and the other for the carry. However, such a polynomial computing the carry is not known for non-binary cases.² This is the main reason why it is hard to extend the circuit above to non-binary message spaces.

1.2 Our Contributions

In this paper, we solve the problem above; for $\mathcal{M} = \mathbb{Z}_Q$ where Q is *any* (constant) prime, we construct an FHE scheme over integers based on a new design principle. We also extend it to a batch FHE scheme with $\mathcal{M} = \mathbb{Z}_{Q_1} \times \dots \times \mathbb{Z}_{Q_k}$, where Q_1, \dots, Q_k may be different. Our main advantages are as follows:

1. The FHE scheme for $Q > 2$ was not achieved in [3, 8].
2. Our decryption circuit has multiplicative degree $O(\lambda)$ for any Q ; even for $Q = 2$, it is significantly improved from $O(\lambda(\log \lambda)^2)$ of [3, 8].

¹ See [8] for how to compute z_i from c and the public key.

² In fact, they exploited the fact that the binary expression of W_j is given by elementary symmetric polynomials in $(s_1 z_{1,j}), \dots, (s_\Theta z_{\Theta,j})$ [1]. However, such an expression is unknown for non-binary cases.

For $\mathcal{M} = \mathbb{Z}_Q$, the encryption is given by $c = pq + Qr + m$. The decryption of the SHE scheme is given by $m = (c \bmod p) \bmod Q = c - p \cdot \lfloor c/p \rfloor \bmod Q$. Then our squashed decryption algorithm works as

$$m \leftarrow c - p \cdot \left\lfloor \sum_{i=1}^{\Theta} s_i z_i \right\rfloor \bmod Q .^3$$

Here, $z_i = (z_{i,0}.z_{i,1} \dots z_{i,L})_Q$ is a real number with $L = \lceil \log_Q \lambda \rceil + 2$ digits of precision after the Q -ary point satisfying $\sum_{i=1}^{\Theta} s_i z_i \approx c/p$.

Now we first determine a polynomial $f(x, y)$ computing the carry of a half adder for any prime Q . Namely, when $x, y \in \mathbb{Z}_Q$ and $x + y = \beta \cdot Q + \alpha \in \mathbb{Z}$, our polynomial f computes $\beta = f(x, y) \bmod Q$.⁴ It has degree Q which is proven to be the lowest. See Sec. 3. Then we compute $\sum_{i=1}^{\Theta} s_i z_i = (w_0.w_1 \dots w_L)_Q \bmod Q$ as follows.⁵

- First, we compute the sum of the last digits as shown in Fig. 1 (where each box is a half adder) so that we obtain w_L and the $\Theta - 1$ carries $\beta_1, \dots, \beta_{\Theta-1}$ with

$$s_1 z_{1,L} + \dots + s_{\Theta} z_{\Theta,L} = w_L + Q \cdot (\beta_1 + \dots + \beta_{\Theta-1}) .$$

- Secondly, we compute $(s_1 z_{1,L-1} + \dots + s_{\Theta} z_{\Theta,L-1}) + (\beta_1 + \dots + \beta_{\Theta-1})$ similarly so that we obtain w_{L-1} and the $2(\Theta - 1)$ carries.
- Iterating this process, we obtain $(w_0.w_1 \dots w_L)_Q$.⁶

The circuit computing each step has multiplicative degree Q ($= \deg f$). Hence, the multiplicative degree D of our decryption circuit is $Q^{L+1} = O(\lambda)$, which is significantly lower than $O(\lambda(\log \lambda)^2)$ of [3, 8].

Finally, in the same way as [3, 8], we make our scheme bootstrappable by letting the bit length of p be $\rho \cdot \Theta(D)$, where ρ is the size of noise r in a fresh ciphertext c . Since the degree D of the decryption circuit has been decreased in comparison to [3, 8], the size of p is also reduced, therefore the size of our ciphertexts is much smaller than that of [3, 8] even for the previously known case $Q = 2$. See Table 1.

Moreover, we emphasize that we also give a *concrete*, not just asymptotic, condition for the parameters of our scheme to make the scheme bootstrappable; see (13).

³ In Sec. 6.1, the information of p is involved in an element X and is reflected by public key components u_ℓ , therefore the decryption does not need p itself.

⁴ One may think that this polynomial would be immediately derived from the Witt polynomials [19], which determine the carry functions in the addition of Q -adic integers. However, Q -adic integers are different from Q -ary expression of integers.

⁵ For the sake of our analysis in Sec. 7, our algorithm in Sec. 4 is described in a different, but essentially equivalent manner.

⁶ We choose the parameters to guarantee that $w_1 \in \{0, Q - 1\}$; consequently $\lfloor \sum_{i=1}^{\Theta} s_i z_i \rfloor \equiv w_0 - w_1 \pmod{Q}$.

Table 1. Bootstrappable Bit Lengths of Secret Prime p

	binary message	non-binary message
DGHV'10 [8], CCK+'13 [3]	$\rho \cdot \Theta(\lambda(\log \lambda)^2)$	—
Our result	$\rho \cdot \Theta(\lambda)$	$\rho \cdot \Theta(\lambda)$

1.3 Organization of the Paper

In Sec. 2, we summarize some definitions and notations used in this paper. In Sec. 3, we study the polynomial expression of the carry function in the addition of two Q -ary digits for any prime Q . Based on the result, in Sec. 4, we construct an algorithm for addition of Q -ary integers which is composed of polynomial evaluations modulo Q . Then, in Sec. 5, we recall the previous SHE; and in Sec. 6, we describe our proposed bootstrapping algorithm based on the result in Sec. 4. Finally, in Sec. 7, we analyze our proposed method to verify that the bootstrapping is indeed achieved.

2 Preliminaries

In this paper, we naturally identify the integer residue ring $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$ modulo an integer $n > 0$ with the set $\{0, 1, \dots, n - 1\}$. For real numbers x, y , we write $x \equiv y \pmod{n}$ if $(x - y)/n \in \mathbb{Z}$. On the other hand, we consider the following two kinds of remainder operations; we define $x \text{ mod } n$ to be the unique $y \in \mathbb{Z}_n$ with $y \equiv x \pmod{n}$, and $x \text{ Mod } n$ to be the unique integer y in $(-n/2, n/2]$ with $y \equiv x \pmod{n}$.

For a Q -ary representation $A = (a_0.a_1, a_2, \dots)_Q$ (with $a_j \in \mathbb{Z}_Q$) of a real number A and an integer $L \geq 0$, we define

$$(A)_L := (a_0.a_1, a_2, \dots, a_L)_Q .$$

For a prime Q , an integer a and an integer $b \in \mathbb{Z}_Q$, we define

$$\binom{a}{b}_Q := a(a - 1) \cdots (a - b + 1) \cdot \text{Inv}_Q(b!) \quad (2)$$

which is a polynomial in a of degree $b \leq Q - 1$, where $\text{Inv}_Q(x)$ (for $x \in \mathbb{Z}$ coprime to Q) denotes the unique integer $y \in \mathbb{Z}_Q$ with $xy \equiv 1 \pmod{Q}$. Then we have

$$\binom{a}{b}_Q \equiv \binom{a}{b} \pmod{Q} \quad (3)$$

(the right-hand side is the usual binomial coefficient).

3 Q-ary Half Adder

Let Q be a prime. For $x, y \in \mathbb{Z}_Q$, let

$$x + y = (c, s)_Q = c \cdot Q + s .$$

It is clear that $s = x + y \bmod Q$. In this section, we construct the lowest degree polynomial $f_{\text{carry},Q}(x, y)$ yielding the carry c , as follows:

Theorem 1. *We define a polynomial $f_{\text{carry},Q}(x, y)$ over the field \mathbb{Z}_Q by*

$$f_{\text{carry},Q}(x, y) := \sum_{i=1}^{Q-1} \binom{x}{i}_Q \binom{y}{Q-i}_Q ,$$

having total degree $\deg f_{\text{carry},Q} = Q$ (see (2) for the notations). Then for any $x, y \in \mathbb{Z}_Q$, we have

$$c = f_{\text{carry},Q}(x, y) \bmod Q .$$

Theorem 2. *The total degree of $f_{\text{carry},Q}$ is lowest among all polynomials $g(x, y)$ over \mathbb{Z}_Q satisfying that $c = g(x, y) \bmod Q$ for any $x, y \in \mathbb{Z}_Q$.*

From now, we prove these two theorems. We note that the first part of the proof of Theorem 1 can be also derived by Lucas' Theorem [14]; here we give a direct proof for the sake of completeness.

Proof (Theorem 1). We first show that, for any $x, y \in \mathbb{Z}_Q$,

$$c = \binom{x+y}{Q} \bmod Q . \quad (4)$$

If $0 \leq x + y < Q$, then we have $c = 0$, while we have $\binom{x+y}{Q} = 0$ by the definition of the binomial coefficient. For the other case $Q \leq x + y < 2Q$, we have $c = 1$, while we have

$$\begin{aligned} \binom{x+y}{Q} &= \binom{x+y}{x+y-Q} \\ &\equiv (x+y)(x+y-1) \cdots (Q+1) \cdot \text{Inv}_Q((x+y-Q) \cdots 1) \\ &\equiv (x+y-Q)(x+y-Q-1) \cdots 1 \cdot \text{Inv}_Q((x+y-Q) \cdots 1) \\ &\equiv 1 \pmod{Q} \end{aligned}$$

(see Sec. 2 for the definition of Inv_Q). Therefore (4) holds.

Next, from the meaning of $\binom{x+y}{Q}$, it is easy to see that

$$\binom{x+y}{Q} = \binom{x}{0} \binom{y}{Q} + \binom{x}{1} \binom{y}{Q-1} + \cdots + \binom{x}{Q} \binom{y}{0} .$$

Now we have $0 \leq x < Q$ and $0 \leq y < Q$ since $x \in \mathbb{Z}_Q$ and $y \in \mathbb{Z}_Q$, therefore $\binom{x}{Q} = \binom{y}{Q} = 0$. Hence, by (3), Theorem 1 holds.

We use the following property in the proof of Theorem 2; we note that the proof of this lemma is similar to the proof of Schwartz–Zippel Theorem [15]:

Lemma 1. *Let $f'(x, y)$ be a polynomial over \mathbb{Z}_Q of degree at most $Q - 1$ with respect to each of x and y . If $f'(x, y) = f_{\text{carry}, Q}(x, y)$ for every $x, y \in \mathbb{Z}_Q$, then f' coincides with $f_{\text{carry}, Q}$ as polynomials.*

Proof. Assume that $f' \neq f_{\text{carry}, Q}$ as polynomials. Set $g := f_{\text{carry}, Q} - f'$, which is now a non-zero polynomial. Write $g(x, y) = \sum_{i=0}^{Q-1} g_i(y)x^i$, where each $g_i(y)$ is a polynomial of degree at most $Q - 1$. Then g_i is a non-zero polynomial for at least one index i . By the polynomial remainder theorem, we have $g_i(b) = 0$ for at most $Q - 1$ elements $b \in \mathbb{Z}_Q$; therefore $g_i(b) \neq 0$ for some $b \in \mathbb{Z}_Q$. Now $g(x, b)$ is a non-zero polynomial in x of degree at most $Q - 1$, therefore $g(a, b) \neq 0$ for some $a \in \mathbb{Z}_Q$ by the same reason. On the other hand, we must have $g(x, y) = f_{\text{carry}, Q}(x, y) - f'(x, y) = 0$ for any $x, y \in \mathbb{Z}_Q$; this is a contradiction. Hence Lemma 1 holds.

Proof (Theorem 2). If such a polynomial $g(x, y)$ has degree at least Q with respect to x (respectively, y), then $\deg g(x, y)$ can be decreased without changing the values $g(x, y) \bmod Q$ by using the relation $x^Q \equiv x \pmod{Q}$ (respectively, $y^Q \equiv y \pmod{Q}$) derived from Fermat’s Little Theorem. Iterating the process, we obtain a polynomial $g_*(x, y)$ of degree at most $Q - 1$ with respect to each of x and y , satisfying that $\deg g_* \leq \deg g$ and $g_*(x, y) \equiv g(x, y) \equiv f_{\text{carry}, Q}(x, y) \pmod{Q}$ for any $x, y \in \mathbb{Z}_Q$. Then Lemma 1 implies that $g_* = f_{\text{carry}, Q}$ as polynomials. Hence we have $\deg f_{\text{carry}, Q} = \deg g_* \leq \deg g$. Therefore Theorem 2 holds.

4 Low-Degree Circuit for Sum of Integers

For $i = 1, \dots, m$, let $\mathbf{a}_i = (a_{i,1}, \dots, a_{i,n})_Q$. In this section, we give a circuit of low (multiplicative) degree which computes

$$\mathbf{a}_1 + \dots + \mathbf{a}_m \bmod Q^n. \quad (5)$$

We call the $m \times n$ matrix $A = (a_{i,j})_{i,j}$ the *matrix representation* of $(\mathbf{a}_1, \dots, \mathbf{a}_m)$.

First we define an algorithm $\text{StreamAdd}_Q(x_1, \dots, x_m)$ for $x_1, \dots, x_m \in \mathbb{Z}_Q$ (see also Fig. 1).

— StreamAdd $_Q(x_1, \dots, x_m)$ —

$$s_2 \leftarrow x_1 + x_2 \bmod Q$$

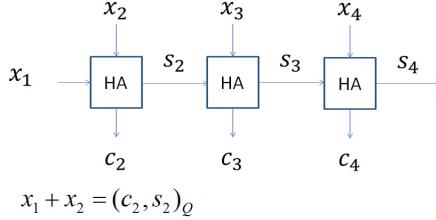
$$c_2 \leftarrow f_{\text{carry}, Q}(x_1, x_2) \bmod Q \quad \% (c_2, s_2)_Q = x_1 + x_2$$

For $i = 3, \dots, m$,

$$s_i \leftarrow s_{i-1} + x_i \bmod Q$$

$$c_i \leftarrow f_{\text{carry}, Q}(s_{i-1}, x_i) \bmod Q \quad \% (c_i, s_i)_Q = s_{i-1} + x_i$$

Return $(s_m, (c_2, \dots, c_m))$

**Fig. 1.** StreamAdd_Q(x_1, \dots, x_4)

For $(s_m, (c_2, \dots, c_m)) \leftarrow \text{StreamAdd}(x_1, \dots, x_m)$, it is easy to see that

$$x_1 + \dots + x_m = s_m + Q \times (c_2 + \dots + c_m) . \quad (6)$$

We next define an algorithm MatrixAdd_Q(A), where $A = (a_{i,j})_{i,j}$ is an $m \times n$ matrix as above.

```

  ┌────────────────────────────────────────────────────────────────────────┐
  | MatrixAddQ( $A$ ) |
  | For  $j = 1, \dots, n$ , |
  |    $(\alpha_j, (\beta_{2,j}, \dots, \beta_{m,j})) \leftarrow \text{StreamAdd}_Q(a_{1,j}, \dots, a_{m,j})$  |
  |   % Apply StreamAddQ to the jth column of  $A$ . |
  | For  $j = 1, \dots, n - 1$ , |
  |    $(b_{1,j}, \dots, b_{m,j}) \leftarrow (\alpha_j, \beta_{2,j+1}, \dots, \beta_{m,j+1})$  |
  |   % Shift  $(\beta_{2,j+1}, \dots, \beta_{m,j+1})^T$  to the left. |
  | Return  $B = (b_{i,j})$  and  $\alpha_n$ , where  $B$  is an  $m \times (n - 1)$  matrix |
  └────────────────────────────────────────────────────────────────────────┘

```

Visually, it can be expressed as

$$B = \begin{pmatrix} \dots, & \boxed{\alpha_{n-1}} \\ \dots, & \boxed{\beta_{2,n}} \\ \vdots & \vdots \\ \dots, & \boxed{\beta_{m,n}} \end{pmatrix}, \boxed{\alpha_n} \leftarrow (\text{StreamAdd}_Q) \leftarrow \begin{pmatrix} \dots, & \boxed{a_{1,n}} \\ \vdots & \vdots \\ \dots, & \boxed{a_{m,n}} \end{pmatrix} = A$$

Given $(B = (b_{i,j}), \alpha_n) \leftarrow \text{MatrixAdd}_Q(A)$, let $\mathbf{b}_i = (b_{i,1}, \dots, b_{i,n-1}, 0)_Q$ for $i = 1, \dots, m$. Then from (6), we can see that

$$\mathbf{a}_1 + \dots + \mathbf{a}_m \equiv (\mathbf{b}_1 + \dots + \mathbf{b}_m) + \alpha_n \pmod{Q^n} .$$

We finally define an algorithm FinalAdd_Q(A), where $A = (a_{i,j})$ is an $m \times n$ matrix.

```

FinalAddQ(A)
Let  $A^{(0)} \leftarrow A$ 
For  $j = 1, \dots, n$ ,
 $(A^{(j)}, d_{n-j+1}) \leftarrow \text{MatrixAdd}_Q(A^{(j-1)})$ ,
where  $A^{(j)}$  is an  $m \times (n-j)$  matrix
Return  $(d_1, \dots, d_n)$ 

```

Suppose that A is the matrix representation of $(\mathbf{a}_1, \dots, \mathbf{a}_m)$. Let

$$(d_1, \dots, d_n) \leftarrow \text{MatrixAdd}_Q(A) .$$

Then it is easy to see that the followings hold (since $\deg f_{\text{carry}, Q} = Q$):

Theorem 3. *We have $(d_1, \dots, d_n)_Q = \mathbf{a}_1 + \dots + \mathbf{a}_m \pmod{Q^n}$.*

Theorem 4. *For $i = 1, \dots, n$, there is a polynomial $f_{Q,i}(x_{1,1}, \dots, x_{m,n})$ over \mathbb{Z}_Q satisfying $\deg f_{Q,i} = Q^{n-i}$ and $d_i = f_{Q,i}(a_{1,1}, \dots, a_{m,n}) \pmod{Q}$.*

5 Batch SHE Scheme over Integers

In this section, we describe an SHE scheme over integers with message space $\mathcal{M} = (\mathbb{Z}_{Q_1})^{h_1} \times \dots \times (\mathbb{Z}_{Q_k})^{h_k}$, where $k \geq 1$, $h_j \geq 1$ and Q_1, \dots, Q_k are distinct primes. This scheme is essentially the one proposed in [3] which is semantically secure under the (ρ, η, γ) -decisional approximate GCD assumption (see [3] for details), with slight notational modifications.⁷⁸ To simplify the notations, set

$$\mathcal{I} := \{(i, j) \mid i, j \in \mathbb{Z}, 1 \leq i \leq k, 1 \leq j \leq h_i\} .$$

The choices of other parameters ρ, γ, η, τ are discussed later.

- Key generation $\text{KeyGen}(1^\lambda)$: Choose η -bit primes $p_{i,j}$ for $(i, j) \in \mathcal{I}$ uniformly at random in a way that all $p_{i,j}$ and $Q_{i'}$ are different. Choose

$$q_0 \stackrel{\$}{\leftarrow} \left[1, 2^\gamma / \prod_{(i,j) \in \mathcal{I}} p_{i,j} \right] \cap \text{ROUGH}(2^{\lambda^2})$$

in a way that q_0 is coprime to all $p_{i,j}$ and all $Q_{i'}$, where $\text{ROUGH}(2^{\lambda^2})$ denotes the set of integers having no prime factors less than 2^{λ^2} . Set

$$N := q_0 \prod_{(i,j) \in \mathcal{I}} p_{i,j} .$$

⁷ In fact, our proposed bootstrapping method is directly extendable to the variant of their scheme in [3] based on the error-free approximate GCD assumption.

⁸ We note that the components of the message space is changed from $(-Q/2, Q/2] \cap \mathbb{Z}$ as in [3] to $\{0, 1, \dots, Q-1\}$, but it does not affect the security of the scheme. Indeed, by the map $c \mapsto c + \sum_{(i,j) \in \mathcal{I}} \lfloor (Q_i - 1)/2 \rfloor x'_{i,j}$ we can convert any ciphertext with the former message space to that with the latter message space, and vice versa.

Choose $e_{\xi;0}$ and $e_{\xi;i,j}$ for $\xi \in \{1, \dots, \tau\}$ and $(i, j) \in \mathcal{I}$ by

$$e_{\xi;0} \xleftarrow{\$} [0, q_0) \cap \mathbb{Z}, e_{\xi;i,j} \xleftarrow{\$} (-2^\rho, 2^\rho) \cap \mathbb{Z}.$$

Then let x_ξ be the unique integer in $(-N/2, N/2]$ satisfying

$$x_\xi \equiv e_{\xi;0} \pmod{q_0}, x_\xi \equiv e_{\xi;i,j} Q_i \pmod{p_{i,j}} \text{ for } (i, j) \in \mathcal{I}.$$

Similarly, for $(i, j), (i', j') \in \mathcal{I}$, choose $e'_{i,j;0}$ and $e_{i,j;i',j'}$ by

$$e'_{i,j;0} \xleftarrow{\$} [0, q_0) \cap \mathbb{Z}, e_{i,j;i',j'} \xleftarrow{\$} (-2^\rho, 2^\rho) \cap \mathbb{Z},$$

and let $x'_{i,j}$ be the unique integer in $(-N/2, N/2]$ satisfying

$$\begin{aligned} x'_{i,j} &\equiv e'_{i,j;0} \pmod{q_0}, \\ x'_{i,j} &\equiv e'_{i,j;i',j'} Q_{i'} + \delta_{(i,j),(i',j')} \pmod{p_{i',j'}} \text{ for } (i', j') \in \mathcal{I}, \end{aligned}$$

where $\delta_{*,*}$ are Kronecker delta. Then output a public key \mathbf{pk} consisting of all N , x_ξ and $x'_{i,j}$, and a secret key \mathbf{sk} consisting of all $p_{i,j}$.

- Encryption $\mathsf{Enc}(\mathbf{pk}, \mathbf{m})$: Given a plaintext $\mathbf{m} = (m_{i,j})_{(i,j) \in \mathcal{I}} \in \mathcal{M}$, output a ciphertext c defined by

$$c := \sum_{(i,j) \in \mathcal{I}} m_{i,j} x'_{i,j} + \sum_{\xi \in T} x_\xi \text{ Mod } N \in (-N/2, N/2] \cap \mathbb{Z},$$

where T is a uniformly random subset of $\{1, 2, \dots, \tau\}$.

- Decryption $\mathsf{Dec}(\mathbf{sk}, c)$: Given a ciphertext c , output $\mathbf{m} \in \mathcal{M}$ given by

$$\mathbf{m} := ((c \text{ Mod } p_{i,j}) \text{ mod } Q_i)_{(i,j) \in \mathcal{I}}.$$

- Evaluation $\mathsf{Eval}(\mathbf{pk}, f, c_1, \dots, c_n)$: Given a polynomial f with integer coefficients and ciphertexts c_1, \dots, c_n , output c^* given by

$$c^* := f(c_1, \dots, c_n) \text{ Mod } N.$$

Following the arguments in [3], we let the parameters ρ , γ , η and τ satisfy the following conditions (see Sec. 6.3 for further details):

- $\rho = \omega(\lambda)$, to resist the attack by Chen and Nguyen [4] for the approximate GCD assumption.
- $\gamma > \eta^2/\rho$, to resist Howgrave-Graham's attack [11] for the approximate GCD assumption.
- $\eta = \Omega(\lambda^2)$ and $\gamma = (\sum_{i=1}^k h_i) \cdot \eta + \Omega(\lambda^2)$, to resist Lenstra's elliptic curve method [13] for factoring the integer N (the latter is to make the (approximate) bit length $\gamma - (\sum_{i=1}^k h_i) \cdot \eta$ of q_0 sufficiently large).
- $\gamma = \eta^2 \omega(\log \lambda)$, to resist the attack by Cohn and Heninger [5] and the attack using Lagarias algorithm [12] on the approximate GCD assumption. (This implies the condition $\gamma = \Omega(\lambda^3)$ arises from the general number field sieve [2] for factoring N .)
- $\tau = \gamma + \omega(\log \lambda)$, in order to use the Leftover Hash Lemma in the security proof (see [3] for the details).

6 Our FHE Scheme: Bootstrapping for Large Plaintexts

We now describe our bootstrapping algorithm for the SHE scheme in Sec. 5 with non-binary plaintexts, based on our results in Sec. 4.

6.1 Squashed Scheme

In this subsection, we squash the decryption algorithm of the SHE scheme in Sec. 5, i.e., we modify the scheme in such a way that the multiplicative degree of the resulting decryption circuit is low enough to make the bootstrapping possible. It is a natural generalization of the squashing method in [8] to the large message space. The choices of additional parameters κ_i , θ_i , Θ_i and L_i for $i \in \{1, \dots, k\}$ will be discussed in Sec. 6.3. Set

$$\Theta_{\max} := \max\{\Theta_1, \dots, \Theta_k\} .$$

From now, we describe the squashed scheme.

- Key Generation $\text{KeyGen}^*(1^\lambda)$: First, generate $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(1^\lambda)$ as in Sec. 5. Then choose a subset Π of the product of symmetric groups $\mathcal{S}_{h_1} \times \dots \times \mathcal{S}_{h_k}$ satisfying that Π contains the identity permutation id and Π generates the group $\mathcal{S}_{h_1} \times \dots \times \mathcal{S}_{h_k}$. Secondly, for each $(i, j) \in \mathcal{I}$, choose uniformly at random a Θ_i -bit vector

$$(s_{i,j;1}, \dots, s_{i,j;\Theta_i}) \in \{0, 1\}^{\Theta_i}$$

with Hamming weight θ_i , and set

$$X_{i,j} := \lfloor Q_i^{\kappa_i} \cdot (p_{i,j} \bmod Q_i) / p_{i,j} \rfloor .$$

For $1 \leq i \leq k$ and $1 \leq \ell \leq \Theta_i$, choose

$$u_{i,\ell} \xleftarrow{\$} [0, Q_i^{\kappa_i+1}) \cap \mathbb{Z}$$

in such a way that

$$\sum_{\ell=1}^{\Theta_i} s_{i,j;\ell} u_{i,\ell} \equiv X_{i,j} \pmod{Q_i^{\kappa_i+1}} \text{ for } 1 \leq j \leq h_i .$$

Moreover, for each $\sigma = (\sigma_1, \dots, \sigma_k) \in \Pi$, generate

$$v_\ell^\sigma \leftarrow \text{Enc}(\mathsf{pk}, \mathbf{m}_\ell^\sigma) \text{ for } 1 \leq \ell \leq \Theta_{\max} ,$$

where $\mathbf{m}_\ell^\sigma = (m_{\ell;i,j}^\sigma)_{(i,j) \in \mathcal{I}} \in \mathcal{M}$ is defined by

$$m_{\ell;i,j}^\sigma = s_{i,\sigma_i(j);\ell} \text{ if } \ell \leq \Theta_i , m_{\ell;i,j}^\sigma = 0 \text{ otherwise.}$$

Then output a public key pk^* consisting of all pk , Π , $u_{i,\ell}$ and v_ℓ^σ , and secret key sk^* consisting of all $s_{i,j;\ell}$.

- Encryption $\text{Enc}^*(\mathbf{pk}^*, \mathbf{m})$ and evaluation $\text{Eval}^*(\mathbf{pk}^*, f, c_1, \dots, c_n)$: These are the same as the scheme in Sec. 5 (with public key \mathbf{pk}).
- Decryption $\text{Dec}^*(\mathbf{sk}^*, c)$: Given a ciphertext c , for $1 \leq i \leq k$ and $1 \leq \ell \leq \Theta_i$, compute

$$z_{i,\ell} := (c \cdot u_{i,\ell} / Q_i^{\kappa_i} \bmod Q_i)_{L_i} .$$

Then output $\mathbf{m} = (m_{i,j})_{(i,j) \in \mathcal{I}}$ defined by

$$m_{i,j} := c - \left[\sum_{\ell=1}^{\Theta_i} s_{i,j;\ell} z_{i,\ell} \right] \bmod Q_i \text{ for each } (i,j) \in \mathcal{I} .$$

We note that, the possible difference of the security of this scheme from the one in Sec. 5 comes from the components $u_{i,\ell}$ involved in the new public key, which are dependent on the secret values $s_{i,j;\ell}$. The situation is the same as the previous FHE schemes [3, 8]. In [8], it was observed that revealing the secret by using the ‘‘hints’’ $u_{i,\ell}$ is related to the sparse subset sum problem and the low-weight knapsack problem. They proposed the following choices of parameters θ_i and Θ_i to avoid the known attacks:

- Θ_i is $\omega(\log \lambda)$ times the bit lengths $(\kappa_i + 1) \log_2 Q_i$ of $u_{i,\ell}$.
- θ_i is large enough to resist brute-force attacks; e.g., $\theta_i := \lambda$ as in [8].

6.2 Our Bootstrapping Procedure

In this subsection, we describe our proposed bootstrapping algorithm based on the results in Sec. 4. More precisely, in the same manner as the previous FHE scheme with modulo-two plaintexts [3], we construct ‘‘permuted bootstrapping’’ algorithm $\text{Bootstrap}(\mathbf{pk}^*, c, \sigma)$ for a ciphertext c for plaintext $(m_{i,j})_{(i,j) \in \mathcal{I}}$ and a permutation $\sigma \in \Pi$, which generates a ciphertext for permuted plaintext $(m_{i,\sigma_i(j)})_{(i,j) \in \mathcal{I}}$ with reduced noise (the case $\sigma = \text{id}$ yields the usual bootstrapping).

Let $\text{StreamAdd}'_Q$ be a variant of the algorithm StreamAdd_Q defined in Sec. 4, obtained in such a way that the inputs are ciphertexts rather than elements of \mathbb{Z}_Q , and the additions and evaluations of the polynomial $f_{\text{carry},Q}$ in StreamAdd_Q modulo Q are replaced with the corresponding homomorphic evaluations for ciphertexts, i.e., additions and evaluations of $f_{\text{carry},Q}$ ‘‘Modulo N ’’. Let $\text{MatrixAdd}'_Q$ and $\text{FinalAdd}'_Q$ be the corresponding variants of MatrixAdd_Q and FinalAdd_Q , respectively. We construct the algorithm $\text{Bootstrap}(\mathbf{pk}^*, c, \sigma)$ by using $\text{FinalAdd}'_Q$, as follows:

- Permutated bootstrapping $\text{Bootstrap}(\mathbf{pk}^*, c, \sigma)$: First, compute $z_{i,\ell}$ for $1 \leq i \leq k$ and $1 \leq \ell \leq \Theta_i$ as in Dec^* , and write

$$z_{i,\ell} = (z_{i,\ell;0}, z_{i,\ell;1}, \dots, z_{i,\ell;L_i})_{Q_i}$$

where $z_{i,\ell;\xi} \in \mathbb{Z}_{Q_i}$ for each $0 \leq \xi \leq L_i$. For $0 \leq \xi \leq L_i$, set

$$v_{i,\ell;\xi} := z_{i,\ell;\xi} \cdot v_\ell^\sigma \bmod N . \quad (7)$$

Then compute

$$(w_{i;0}, w_{i;1}, \dots, w_{i;L_i}) \leftarrow \text{FinalAdd}'_{Q_i}(V_i) ,$$

where $V_i = (v_{i,\ell;\xi})_{1 \leq \ell \leq \Theta_i, 0 \leq \xi \leq L_i}$ is a $\Theta_i \times (L_i + 1)$ matrix consisting of ciphertexts. Moreover, for $1 \leq i \leq k$, compute

$$c^{\langle i \rangle} \leftarrow \text{crt}_i \cdot (w_{i;0} - w_{i;1}) \text{ Mod } N ,$$

where crt_i denotes the unique integer in $(-(\prod_{i'=1}^k Q_{i'})/2, (\prod_{i'=1}^k Q_{i'})/2]$ with $\text{crt}_i \equiv \delta_{i,i'} \pmod{Q_{i'}}$ for any $1 \leq i' \leq k$. Finally, output

$$c^* \leftarrow \left(c \text{ Mod } \prod_{i=1}^k Q_i \right) - c^{\langle 1 \rangle} - \dots - c^{\langle k \rangle} \text{ Mod } N .$$

From Theorem 4, the multiplicative degree of the $\text{FinalAdd}'$ circuit is

$$Q_i^{L_i+1} = Q_i^{\lceil \log_{Q_i} \lambda \rceil + 3} \leq Q_i^{\log_{Q_i} \lambda + 4} = Q_i^4 \cdot \lambda$$

(see Sec. 6.3 below for the choice of L_i) which is $O(\lambda)$ for a constant Q_i .

6.3 Choice of Parameters

We give an instance of the choice of parameters for our scheme, where we regard all of k , Q_i and h_i as constants. First, we set

$$\rho = \Theta(\lambda \log \log \log \lambda), \eta = \Theta(\lambda^2 \log \log \lambda), \gamma = \Theta(\lambda^4 (\log \lambda)^2), \tau = \gamma + \lambda .$$

Then all the conditions mentioned in Sec. 5 are indeed satisfied. Secondly, for the additional parameters in the squashed scheme, we set

$$\begin{aligned} L_i &= \lceil \log_{Q_i} \theta_i \rceil + 2, \quad \kappa_i = \lceil (\gamma - \log_2(4Q_i - 5)) / \log_2 Q_i \rceil + 2 , \\ \Theta_i &= \Theta((\lambda \log \lambda)^4), \quad \theta_i = \lambda \text{ for each } 1 \leq i \leq k . \end{aligned} \tag{8}$$

Then the conditions mentioned in Sec. 6.1 are also satisfied. Moreover, the analysis given in Sec. 7 below shows that our scheme is indeed bootstrappable by using these parameters. We emphasize that the order of the bit length η of $p_{i,j}$ is only slightly higher than $\rho \cdot \lambda$, which is significantly lower than $\rho \cdot \lambda (\log \lambda)^2$ required in the previous FHE schemes [3, 8] (see also (13) in Sec. 7 for a concrete lower bound for η). This reduces the key sizes for the scheme, even in the previously achieved cases $Q_i = 2$.

7 Analysis of Our Proposed Scheme

In this section, we analyze our proposed scheme, especially our bootstrapping algorithm, to see the correctness of the scheme and estimate appropriate parameters. For the purpose, we introduce the following definition, which intuitively means the amount of noise in a ciphertext:

Definition 1. Let c be a ciphertext for plaintext $\mathbf{m} = (m_{i,j})_{(i,j) \in \mathcal{I}}$. We define the weight $\text{wt}_{i,j}(c)$ of c at position $(i, j) \in \mathcal{I}$ to be the minimum integer satisfying the following for some $\alpha_{i,j}(c), \beta_{i,j}(c) \in \mathbb{Z}$:

$$c = \alpha_{i,j}(c) \cdot p_{i,j} + \beta_{i,j}(c) \cdot Q_i + m_{i,j} \text{ and } |\beta_{i,j}(c) \cdot Q_i + m_{i,j}| \leq \text{wt}_{i,j}(c) .$$

We evaluate the weights of fresh ciphertexts:

Proposition 1. For any $c \leftarrow \text{Enc}^*(\text{pk}^*, \mathbf{m})$ with $\mathbf{m} \in \mathcal{M}$, we have $\text{wt}_{i,j}(c) \leq Q_i \Gamma$ for any $(i, j) \in \mathcal{I}$, where we define

$$\Gamma := (h_1(Q_1 - 1) + \dots + h_k(Q_k - 1) + \tau) \cdot 2^\rho .$$

Proof. For $(i, j) \in \mathcal{I}$, since N is a multiple of $p_{i,j}$, we have

$$\begin{aligned} c &\equiv \sum_{(i',j') \in \mathcal{I}} m_{i',j'}(e'_{i',j';i,j}Q_i + \delta_{(i',j'),(i,j)}) + \sum_{\xi \in T} e_{\xi;i,j}Q_i \\ &\equiv \left(\sum_{(i',j') \in \mathcal{I}} m_{i',j'}e'_{i',j';i,j} + \sum_{\xi \in T} e_{\xi;i,j} \right) Q_i + m_{i,j} \pmod{p_{i,j}} . \end{aligned}$$

Since $m_{i',j'} \in [0, Q_{i'} - 1]$ and $e_{\xi;i,j}, e'_{i',j';i,j} \in (-2^\rho, 2^\rho)$, the absolute value of the right-hand side is bounded by

$$\left(\sum_{(i',j') \in \mathcal{I}} (Q_{i'} - 1) + \tau \right) (2^\rho - 1)Q_i + Q_i - 1 \leq \left(\sum_{i'=1}^k h_{i'}(Q_{i'} - 1) + \tau \right) Q_i 2^\rho .$$

Hence we have $\text{wt}_{i,j}(c) \leq Q_i \Gamma$, therefore Proposition 1 holds.

The next property is implied directly by the definition of $\text{wt}_{i,j}(c)$:

Proposition 2. Let c_ℓ be a ciphertext for plaintext $(m_{\ell;i,j})_{(i,j) \in \mathcal{I}}$, $1 \leq \ell \leq n$. Let f be a polynomial, and let f_{abs} denote the polynomial obtained by replacing the coefficients in f with their absolute values. Then the output $c \leftarrow \text{Eval}^*(\text{pk}^*, f, c_1, \dots, c_\ell)$ of the evaluation algorithm is a ciphertext for plaintext $(f(m_{1;i,j}, \dots, m_{\ell;i,j}) \bmod Q_i)_{i,j} \in \mathcal{M}$, and we have

$$\text{wt}_{i,j}(c) \leq f_{\text{abs}}(\text{wt}_{i,j}(c_1), \dots, \text{wt}_{i,j}(c_\ell)) \text{ for any } (i, j) \in \mathcal{I} .$$

From now, we show the correctness of the squashed scheme:

Lemma 2. Let c be a ciphertext for plaintext $\mathbf{m} = (m_{i,j})_{(i,j) \in \mathcal{I}}$ with weight $\text{wt}_{i,j}(c)$. Then for any $(i, j) \in \mathcal{I}$, we have

$$\sum_{\ell=1}^{\Theta_i} s_{i,j;\ell} z_{i,\ell} = p_{i,j} \cdot \alpha_{i,j}(c) + \tilde{\beta}_{i,j} \cdot Q_i + \varepsilon_{i,j} \tag{9}$$

for the integer $\alpha_{i,j}(c)$ in Definition 1, some integer $\tilde{\beta}_{i,j}$ and some value $\varepsilon_{i,j}$ with $|\varepsilon_{i,j}| \leq \tilde{\varepsilon}_{i,j}(c)$, where

$$\tilde{\varepsilon}_{i,j}(c) := Q_i \text{wt}_{i,j}(c)/(2p_{i,j}) + \theta_i \cdot Q_i^{-L_i} + NQ_i^{-\kappa_i}/4 .$$

Proof. First, by the definition of $z_{i,\ell}$, we have $z_{i,\ell} = c \cdot u_{i,\ell}/Q_i^{\kappa_i} + A_{i,\ell}Q_i + \Delta_{i,\ell}$ for some $\Delta_{i,\ell}$ with $|\Delta_{i,\ell}| < Q_i^{-L_i}$ and an integer $A_{i,\ell}$. Then we have

$$\sum_{\ell=1}^{\Theta_i} s_{i,j;\ell} z_{i,\ell} = c \cdot \sum_{\ell=1}^{\Theta_i} s_{i,j;\ell} u_{i,\ell}/Q_i^{\kappa_i} + A'_{i,j}Q_i + \Delta'_{i,j} \quad (10)$$

for some $A'_{i,j} \in \mathbb{Z}$ and $\Delta'_{i,j} := \sum_{\ell=1}^{\Theta_i} s_{i,j;\ell} \Delta_{i,\ell}$ with $|\Delta'_{i,j}| < \theta_i \cdot Q_i^{-L_i}$ (recall that $(s_{i,j;1}, \dots, s_{i,j;\Theta_i})$ has Hamming weight θ_i). Now, by the definitions of $X_{i,j}$ and $u_{i,\ell}$, there are an integer $B_{i,j}$ and a value $\Delta''_{i,j} \in [-1/2, 1/2]$ satisfying that the right-hand side of (10) is equal to

$$\begin{aligned} & c \cdot (Q_i^{\kappa_i} \cdot (p_{i,j} \text{ Mod } Q_i)/p_{i,j} + \Delta''_{i,j} + B_{i,j}Q_i^{\kappa_i+1})/Q_i^{\kappa_i} + A'_{i,j}Q_i + \Delta'_{i,j} \\ &= (p_{i,j} \text{ Mod } Q_i) \cdot c/p_{i,j} + (cB_{i,j} + A'_{i,j})Q_i + \Delta'_{i,j} + \Delta''_{i,j}c/Q_i^{\kappa_i} . \end{aligned}$$

Moreover, by the expression of c as in Definition 1, the right-hand side above is equal to the right-hand side of (9), where

$$\begin{aligned} \tilde{\beta}_{i,j} &:= cB_{i,j} + A'_{i,j} - \frac{p_{i,j} - (p_{i,j} \text{ Mod } Q_i)}{Q_i} \cdot \alpha_{i,j}(c) \in \mathbb{Z} , \\ \varepsilon_{i,j} &:= (p_{i,j} \text{ Mod } Q_i)(\beta_{i,j}(c) \cdot Q_i + m_{i,j})/p_{i,j} + \Delta'_{i,j} + \Delta''_{i,j}c/Q_i^{\kappa_i} . \end{aligned}$$

Now, by the definition of $\text{wt}_{i,j}$, we have $|\beta_{i,j}(c) \cdot Q_i + m_{i,j}| \leq \text{wt}_{i,j}(c)$ and

$$|\varepsilon_{i,j}| \leq (Q_i/2) \cdot \text{wt}_{i,j}(c)/p_{i,j} + \theta_i \cdot Q_i^{-L_i} + (1/2) \cdot (N/2)/Q_i^{\kappa_i} = \tilde{\varepsilon}_{i,j}(c) .$$

Hence, Lemma 2 holds.

Proposition 3. *Let c be as in Lemma 2. If $\tilde{\varepsilon}_{i,j}(c) < 1/2$ for any $(i, j) \in \mathcal{I}$ (see Lemma 2 for the definition), then $\text{Dec}^*(\text{sk}^*, c)$ outputs \mathbf{m} correctly.*

Proof. Recall the result of Lemma 2. Then for $(i, j) \in \mathcal{I}$, we have $|\varepsilon_{i,j}| < 1/2$. Therefore, by Definition 1, we have

$$c - \left\lfloor \sum_{\ell=1}^{\Theta_i} s_{i,j;\ell} z_{i,\ell} \right\rfloor \equiv c - p_{i,j} \cdot \alpha_{i,j}(c) \equiv m_{i,j} \pmod{Q_i} .$$

Hence, Proposition 3 holds.

From now, in order to analyze our bootstrapping algorithm, we consider the following condition for ciphertexts which is in general stronger than the condition mentioned in Proposition 3 for correct decryption:

Definition 2. *We say that a ciphertext c is bootstrappable, if $\tilde{\varepsilon}_{i,j}(c) < 1/Q_i$ for any $(i, j) \in \mathcal{I}$ (see Lemma 2 for the definition of $\tilde{\varepsilon}_{i,j}(c)$).*

We analyze the algorithm $\text{Bootstrap}(\text{pk}^*, c, \sigma)$. We assume that c is bootstrappable. For any ciphertext c' , let $m(c') = (m(c')_{i,j})_{(i,j) \in \mathcal{I}}$ denote the plaintext for c' . Set

$$w'_i := w_{i;0} - w_{i;1} \text{ Mod } N \text{ for any } 1 \leq i \leq k .$$

We analyze $\text{FinalAdd}'_{Q_i}(V_i)$ for $1 \leq i \leq k$. First, for $1 \leq j \leq h_i$, we have $m(v_{i,\ell;\xi})_{i,j} = s_{i,\sigma_i(j);\ell} z_{i,\ell;\xi}$ for each ℓ, ξ . Therefore, by Theorem 3 (applied to the L_i -digit shift of the sum of $s_{i,\sigma_i(j);\ell} z_{i,\ell}$ to the left), we have

$$\begin{aligned} & (m(w_{i;0})_{i,j} \cdot m(w_{i;1})_{i,j}, \dots, m(w_{i;L_i})_{i,j})_{Q_i} \\ & \equiv \sum_{\ell=1}^{\Theta_i} s_{i,\sigma_i(j);\ell} z_{i,\ell} \equiv p_{i,\sigma_i(j)} \cdot \alpha_{i,\sigma_i(j)}(c) + \varepsilon_{i,\sigma_i(j)} \pmod{Q_i} \end{aligned}$$

(see Lemma 2 for the last relation). By Lemma 2, we have $|\varepsilon_{i,\sigma_i(j)}| < 1/Q_i$ since c is bootstrappable. Therefore, one of the followings holds:

$$\begin{cases} m(w_{i;1})_{i,j} = 0 \text{ and } p_{i,\sigma_i(j)} \cdot \alpha_{i,\sigma_i(j)}(c) \equiv m(w_{i;0})_{i,j} \pmod{Q_i}, \\ m(w_{i;1})_{i,j} = Q_i - 1 \text{ and } p_{i,\sigma_i(j)} \cdot \alpha_{i,\sigma_i(j)}(c) \equiv m(w_{i;0})_{i,j} + 1 \pmod{Q_i}. \end{cases}$$

In any case, we have

$$m(w'_i)_{i,j} \equiv m(w_{i;0})_{i,j} - m(w_{i;1})_{i,j} \equiv p_{i,\sigma_i(j)} \cdot \alpha_{i,\sigma_i(j)}(c) \pmod{Q_i}.$$

On the other hand, Definition 1 applied to ciphertexts $w'_{i'}$ implies that

$$c^* = p_{i,j} \alpha_{i,j}(c^*) + (c \text{ Mod } Q_1 \cdots Q_k) - \sum_{i'=1}^k \text{crt}_{i'}(\beta_{i,j}(w'_{i'}) \cdot Q_i + m(w'_{i'})_{i,j}),$$

where $\alpha_{i,j}(c^*) = -\sum_{i'=1}^k \text{crt}_{i'} \cdot \alpha_{i,j}(w'_{i'})$. Now, by the definition of $\text{crt}_{i'}$,

$$\begin{aligned} & (c \text{ Mod } Q_1 \cdots Q_k) - \sum_{i'=1}^k \text{crt}_{i'}(\beta_{i,j}(w'_{i'}) \cdot Q_i + m(w'_{i'})_{i,j}) \\ & \equiv c - m(w'_i)_{i,j} \equiv c - p_{i,\sigma_i(j)} \cdot \alpha_{i,\sigma_i(j)}(c) \equiv m(c)_{i,\sigma_i(j)} \pmod{Q_i} \end{aligned}$$

(note that $c = p_{i,\sigma_i(j)} \cdot \alpha_{i,\sigma_i(j)}(c) + \beta_{i,\sigma_i(j)} \cdot Q_i + m(c)_{i,\sigma_i(j)}$ by Definition 1). Therefore, c^* is a ciphertext for plaintext $(m_{i,\sigma_i(j)}(c))_{(i,j) \in \mathcal{I}}$, with weights satisfying the following (since $|\text{crt}_{i'}| \leq Q_1 \cdots Q_k/2$):

$$\begin{aligned} \text{wt}_{i,j}(c^*) & \leq \left| (c \text{ Mod } Q_1 \cdots Q_k) - \sum_{i'=1}^k \text{crt}_{i'}(\beta_{i,j}(w'_{i'}) \cdot Q_i + m(w'_{i'})_{i,j}) \right| \\ & \leq \frac{Q_1 \cdots Q_k}{2} \left(1 + \sum_{i'=1}^k \text{wt}_{i,j}(w'_{i'}) \right) \\ & \leq \frac{Q_1 \cdots Q_k}{2} \left(1 + \sum_{i'=1}^k (\text{wt}_{i,j}(w'_{i';0}) + \text{wt}_{i,j}(w'_{i';1})) \right). \end{aligned}$$

From now, we evaluate the weights $\text{wt}_{i,j}(w'_{i';0})$ and $\text{wt}_{i,j}(w'_{i';1})$:

Lemma 3. Let $(i, j) \in \mathcal{I}$. For two ciphertexts c_1, c_2 , suppose that $\text{wt}_{i,j}(c_1) \leq \alpha$ and $\text{wt}_{i,j}(c_2) \leq \beta$, where $1 < \beta < \alpha$. Then we have

$$\text{wt}_{i,j}(f_{\text{carry},Q}(c_1, c_2)) \leq \lfloor Q/2 \rfloor \cdot \frac{\beta}{\beta - 1} \cdot \frac{(\alpha/\beta)}{(\alpha/\beta) - 1} \cdot \alpha^{Q-1} \beta .$$

Proof. First, each monomial in $f_{\text{carry},Q}(t_1, t_2)$ is of the form $at_1^{d_1}t_2^{d_2}$ with $|a| \leq Q/2$, $d_1, d_2 \in \{1, 2, \dots, Q-1\}$ and $d_1 + d_2 \leq Q$. Therefore, by Proposition 2, we have

$$\text{wt}_{i,j}(f_{\text{carry},Q}(c_1, c_2)) \leq \lfloor Q/2 \rfloor \sum_{\substack{d_1, d_2 \in \{1, 2, \dots, Q-1\} \\ d_1 + d_2 \leq Q}} \alpha^{d_1} \beta^{d_2} .$$

Now the sum in the right-hand side is

$$\sum_{d_1=1}^{Q-1} \alpha^{d_1} \sum_{d_2=1}^{Q-d_1} \beta^{d_2} = \sum_{d_1=1}^{Q-1} \alpha^{d_1} \frac{\beta}{\beta - 1} (\beta^{Q-d_1} - 1) \leq \frac{\beta}{\beta - 1} \sum_{d_1=1}^{Q-1} \alpha^{d_1} \beta^{Q-d_1}$$

(where we used the relation $\beta > 1$), and similarly, the sum in the right-hand side above is

$$\alpha \beta^{Q-1} \sum_{d_1=0}^{Q-2} (\alpha/\beta)^{d_1} \leq \alpha \beta^{Q-1} \cdot \frac{(\alpha/\beta)^{Q-1}}{(\alpha/\beta) - 1} = \alpha^{Q-1} \beta \cdot \frac{(\alpha/\beta)}{(\alpha/\beta) - 1}$$

(where we used the relation $\alpha/\beta > 1$). Hence, Lemma 3 holds.

Lemma 4. Let $A = (a_{\ell,\xi})_{\ell,\xi}$ be a matrix of ciphertexts $a_{\ell,\xi}$ with Θ rows. Let $\mu > 1$. For each $(i, j) \in \mathcal{I}$, if $\text{wt}_{i,j}(a_{\ell,\xi}) \leq \mu$ for any ℓ, ξ , then the output $((b_{\ell,\xi})_{\ell,\xi}, d)$ of $\text{MatrixAdd}'_Q(A)$ satisfies that $\text{wt}_{i,j}(d) \leq \Theta \cdot \mu$ and

$$\text{wt}_{i,j}(b_{\ell,\xi}) \leq \lfloor Q/2 \rfloor \cdot \frac{\mu}{\mu - 1} \cdot \frac{\Theta}{\Theta - 1} \cdot \Theta^{Q-1} \mu^Q \text{ for any } \ell, \xi . \quad (11)$$

Proof. For the intermediate objects s_ξ and c_ξ in the subroutine $\text{StreamAdd}'_Q$ whose inputs are Θ components of A , we have $\text{wt}_{i,j}(s_\xi) \leq \Theta \mu$ for any ξ by the choice of μ , while Lemma 3 with $\alpha := \Theta \mu$ and $\beta := \mu$ implies that $\text{wt}_{i,j}(c_\xi)$ is bounded by the right-hand side of (11). Hence, Lemma 4 holds (note that the right-hand side of (11) is larger than $\Theta \mu$).

Lemma 5. Let $(i, j) \in \mathcal{I}$ and $1 \leq i' \leq k$. For $\xi = 1, \dots, L_{i'} + 1$, let $(V^{(\xi)}, w_{i';L_{i'}+1-\xi})$ denote the output of the subroutine $\text{MatrixAdd}'_{Q_{i'}}(V^{(\xi-1)})$ in $\text{FinalAdd}'_{Q_{i'}}(V_{i'})$, where $V^{(0)} := V_{i'}$. Then we have $\text{wt}_{i,j}(w_{i';L_{i'}+1-\xi}) \leq (\Xi_{i,i'})^{Q_{i'} \xi^{-1}}$, where

$$\Xi_{i,i'} := \left(\left\lfloor \frac{Q_{i'}}{2} \right\rfloor \frac{Q_{i'} Q_i \Gamma}{Q_{i'} Q_i \Gamma - 1} \cdot \frac{\Theta_{i'}}{\Theta_{i'} - 1} \right)^{(Q_{i'} - 1)^{-1}} \cdot \Theta_{i'} Q_{i'} Q_i \Gamma$$

(see Proposition 1 for the definition of Γ).

Proof. We show that $\text{wt}_{i,j}(v^{(\xi)}) \leq \mu_\xi / \Theta_{i'}$ for any $0 \leq \xi \leq L_{i'} + 1$ and any component $v^{(\xi)}$ of $V^{(\xi)}$, where

$$\mu_\xi := \left(\left\lfloor \frac{Q_{i'}}{2} \right\rfloor \frac{Q_{i'} Q_i \Gamma}{Q_{i'} Q_i \Gamma - 1} \cdot \frac{\Theta_{i'}}{\Theta_{i'} - 1} \right)^{1+Q_{i'}+\dots+Q_{i'}^{\xi-1}} \cdot (\Theta_{i'} Q_{i'} Q_i \Gamma)^{Q_{i'}^\xi} .$$

Once this is shown, we have $\text{wt}_{i,j}(w_{i';L_{i'}+1-\xi}) \leq \mu_{\xi-1}$ by Lemma 4, while we have $\mu_\xi \leq (\Xi_{i,i'})^{Q_{i'}^\xi}$ since $1 + Q_{i'} + \dots + Q_{i'}^{\xi-1} \leq Q_{i'}^\xi / (Q_{i'} - 1)$, therefore the claim will follow.

First, for $\xi = 0$, we have $\mu_0 / \Theta_{i'} = Q_{i'} Q_i \Gamma$, while we have $\text{wt}_{i,j}(v^{(0)}) \leq Q_{i'} Q_i \Gamma$ by (7) and Proposition 1. Hence, the claim holds for the case.

For $\xi > 0$, we use the induction on ξ . First, we have

$$\frac{\mu_{\xi-1} / \Theta_{i'}}{\mu_{\xi-1} / \Theta_{i'} - 1} \leq \frac{Q_{i'} Q_i \Gamma}{Q_{i'} Q_i \Gamma - 1}$$

since $\mu_{\xi-1} / \Theta_{i'} \geq \mu_0 / \Theta_{i'} = Q_{i'} Q_i \Gamma$. Then by Lemma 4 and the induction hypothesis, we have

$$\text{wt}_{i,j}(v^{(\xi)}) \leq \left\lfloor \frac{Q_{i'}}{2} \right\rfloor \frac{Q_{i'} Q_i \Gamma}{Q_{i'} Q_i \Gamma - 1} \cdot \frac{\Theta_{i'}}{\Theta_{i'} - 1} \cdot \Theta_{i'}^{Q_{i'} - 1} \left(\frac{\mu_{\xi-1}}{\Theta_{i'}} \right)^{Q_{i'}} = \frac{\mu_\xi}{\Theta_{i'}} .$$

Hence the claim holds for the case, therefore Lemma 5 holds.

By Lemma 5, we have

$$\begin{aligned} \text{wt}_{i,j}(c^*) &\leq \frac{Q_1 \cdots Q_k}{2} \left(1 + \sum_{i'=1}^k ((\Xi_{i,i'})^{Q_{i'} L_{i'}} + (\Xi_{i,i'})^{Q_{i'} L_{i'} - 1}) \right) \\ &\leq \frac{Q_1 \cdots Q_k}{2} \sum_{i'=1}^k (1 + (\Xi_{i,i'})^{Q_{i'} L_{i'}} + (\Xi_{i,i'})^{Q_{i'} L_{i'} - 1}) \leq Q_1 \cdots Q_k \sum_{i'=1}^k (\Xi_{i,i'})^{Q_{i'} L_{i'}} \end{aligned}$$

where we used the relation

$$1 + (\Xi_{i,i'})^{Q_{i'} L_{i'} - 1} \leq \left((\Xi_{i,i'})^{Q_{i'} L_{i'} - 1} \right)^2 \leq \left((\Xi_{i,i'})^{Q_{i'} L_{i'} - 1} \right)^{Q_{i'}}$$

(note that $(\Xi_{i,i'})^{Q_{i'} L_{i'} - 1} \geq 2$). Summarizing, we have the following result:

Theorem 5. Suppose that the parameters satisfy

$$Q_i \cdot Q_1 \cdots Q_k \sum_{i'=1}^k (\Xi_{i,i'})^{Q_{i'} L_{i'}} / (2p_{i,j}) + \theta_i \cdot Q_i^{-L_i} + NQ_i^{-\kappa_i} / 4 < 1/Q_i \quad (12)$$

for any $(i, j) \in \mathcal{I}$ (see Lemma 5 for the definition of $\Xi_{i,i'}$). Then, for any ciphertext c for plaintext $(m_{i,j})_{(i,j) \in \mathcal{I}}$ which is bootstrappable in the sense of Definition 2 and any $\sigma \in \Pi$, the output $c^* \leftarrow \text{Bootstrap}(\text{pk}^*, c, \sigma)$ is a ciphertext for plaintext $(m_{i,\sigma_i(j)})_{(i,j) \in \mathcal{I}}$ which is bootstrappable.

Finally, we investigate the choice of parameters to satisfy the condition (12). First, for the parameters L_i and κ_i in (8), we have

$$\theta_i \cdot Q_i^{-L_i} + NQ_i^{-\kappa_i}/4 \leq 1/Q_i^2 + (4Q_i - 5)/(4Q_i^2) = 1/Q_i - 1/(4Q_i^2)$$

since $N \leq 2^\gamma$, while we have $Q_{i'}^{L_{i'}} \leq \theta_{i'} Q_{i'}^3$. On the other hand, we have $p_{i,j} \geq 2^{\eta-1}$ since $p_{i,j}$ is an η -bit prime. Therefore, to satisfy (12), it suffices to satisfy the following (where we used $\theta_{i'} = \lambda$ as in Sec. 6.3):

$$Q_i \cdot Q_1 \cdots Q_k \sum_{i'=1}^k (\Xi_{i,i'})^{\lambda Q_{i'}^3} / 2^\eta \leq 1/(4Q_i^2) ,$$

or, more strongly,

$$\eta \geq 2 + \log_2(Q_i^3 \cdot Q_1 \cdots Q_k \cdot k) + \lambda \max_{1 \leq i' \leq k} Q_{i'}^3 \log_2 \Xi_{i,i'} . \quad (13)$$

From now, we study the asymptotic behavior of the parameters. By using the relation $t/(t-1) \leq e^{(t-1)^{-1}}$ for $t > 1$, we have

$$\Xi_{i,i'} \leq \left(\frac{Q_{i'}}{2} \cdot e^{(Q_{i'} Q_i \Gamma - 1)^{-1} + (\Theta_{i'} - 1)^{-1}} \right)^{(Q_{i'} - 1)^{-1}} \Theta_{i'} Q_{i'} Q_i \Gamma ,$$

therefore

$$\begin{aligned} \log_2 \Xi_{i,i'} &\leq \frac{1}{Q_{i'} - 1} \left(\log_2 Q_{i'} - 1 + \left(\frac{1}{Q_{i'} Q_i \Gamma - 1} + \frac{1}{\Theta_{i'} - 1} \right) \log_2 e \right) \\ &\quad + \log_2 \Theta_{i'} + \log_2 Q_{i'} + \log_2 Q_i + \log_2 \Gamma . \end{aligned}$$

Moreover, we have

$$\log_2 \Gamma = \rho + \log_2(h_1(Q_1 - 1) + \cdots + h_k(Q_k - 1) + \tau) .$$

Now, for the choice of parameters in Sec. 6.3, the term ρ in $\log_2 \Gamma$ is dominant among the terms in the upper bound for $\log_2 \Xi_{i,i'}$ above, therefore it suffices to set $\eta = \omega(\rho \cdot \lambda)$ to satisfy (13) asymptotically. Hence, the choice of parameters in Sec. 6.3 is suitable to enable the bootstrapping.

Acknowledgments. The authors thank Shizuo Kaji, Toshiaki Maeno and Yasuhide Numata for their valuable comments on this work, and thank the members of Shin-Akarui-Angou-Benkyo-Kai for discussions on this work. The authors also thank the anonymous reviewers for their precious comments.

References

- Boyar, J., Peralta, R., Pochuev, D.: On the Multiplicative Complexity of Boolean Functions over the Basis $(\wedge, \oplus, 1)$. *Theor. Comput. Sci.* **235**(1), 43–57 (2000)

2. Buhler, J., Lenstra Jr., H.W., Pomerance, C.: Factoring integers with the number field sieve. In: Lenstra, A., Jr. Lenstra, H.W. (eds.) *The Development of the Number Field Sieve*. Lecture Notes in Mathematics, vol. 1554, pp. 50–94. Springer (1993)
3. Cheon, J.H., Coron, J.-S., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch fully homomorphic encryption over the integers. In: Johansson, T., Nguyen, P.Q. (eds.) *EUROCRYPT 2013*. LNCS, vol. 7881, pp. 315–335. Springer, Heidelberg (2013)
4. Chen, Y., Nguyen, P.Q.: Faster algorithms for approximate common divisors: breaking fully-homomorphic-encryption challenges over the integers. In: Pointcheval, D., Johansson, T. (eds.) *EUROCRYPT 2012*. LNCS, vol. 7237, pp. 502–519. Springer, Heidelberg (2012)
5. Cohn, H., Heninger, N.: Approximate Common Divisors via Lattices. IACR Cryptology ePrint Archive 2011/437 (2011)
6. Coron, J.-S., Lepoint, T., Tibouchi, M.: Scale-invariant fully homomorphic encryption over the integers. In: Krawczyk, H. (ed.) *PKC 2014*. LNCS, vol. 8383, pp. 311–328. Springer, Heidelberg (2014)
7. Coron, J.-S., Mandal, A., Naccache, D., Tibouchi, M.: Fully homomorphic encryption over the integers with shorter public keys. In: Rogaway, P. (ed.) *CRYPTO 2011*. LNCS, vol. 6841, pp. 487–504. Springer, Heidelberg (2011)
8. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) *EUROCRYPT 2010*. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
9. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC 2009, pp. 169–178 (2009)
10. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) *PKC 2012*. LNCS, vol. 7293, pp. 1–16. Springer, Heidelberg (2012)
11. Howgrave-Graham, N.: Approximate integer common divisors. In: Silverman, J.H. (ed.) *CaLC 2001*. LNCS, vol. 2146, pp. 51–66. Springer, Heidelberg (2001)
12. Lagarias, J.C.: The Computational Complexity of Simultaneous Diophantine Approximation Problems. *SIAM J. Comput.* **14**(1), 196–209 (1985)
13. Lenstra Jr., H.W.: Factoring Integers with Elliptic Curves. *Annals of Math.*, Second Series **126**(3), 649–673 (1987)
14. Lucas, E.: Théorie des fonctions numériques simplement périodiques. *Amer. J. Math.* **1**(3), 197–240 (1878)
15. Schwartz, J.: Fast probabilistic algorithms for verification of polynomial identities. *J. ACM* **27**, 701–717 (1980)
16. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) *PKC 2010*. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010)
17. Smart, N.P., Vercauteren, F.: Fully Homomorphic SIMD Operations. *Des. Codes Cryptography* **71**(1), 57–81 (2014)
18. Stanley, R. P.: *Enumerative Combinatorics*, 1st edn. vol. I. Cambridge University Press (1997)
19. Witt vector. <http://en.wikipedia.org/wiki/Wittvector>

Related-Key Attacks

KDM-CCA Security from RKA Secure Authenticated Encryption

Xianhui Lu^{1,2(✉)}, Bao Li^{1,2}, and Dingding Jia^{1,2}

¹ Data Assurance and Communication Security Research Center,
Chinese Academy of Sciences, Beijing 10093, China

² State Key Laboratory of Information Security, Institute of Information
Engineering, Chinese Academy of Sciences, Beijing 100093, China
{xhlu,lb,ddj}@is.ac.cn

Abstract. We propose an efficient public key encryption scheme which is key-dependent message secure against chosen ciphertext attacks (KDM-CCA) with respect to affine functions based on the decisional composite residuosity assumption. Technically, we achieve KDM-CCA security by enhancing a chosen ciphertext secure scheme based on the high entropy hash proof system with three tools: a key-dependent message encoding, an entropy filter and an authenticated encryption secure against related-key attacks.

Keywords: Public key encryption · Key-dependent message security · Related key attack · Authenticated encryption

1 Introduction

An encryption scheme is key-dependent message (KDM) secure if it is secure even when the adversary can access encryptions of messages that depend on the secret key. Due to its extensive usefulness in cryptographic protocol design and analysis [16, 23], hard disk encryption[19] and fully homomorphic public key encryption [28], KDM security was widely studied in recent years [2–4, 7, 8, 11, 17, 19–21, 31, 32, 34, 40].

Although the construction of KDM secure schemes in the random oracle model is very easy [6, 16, 23], in the standard model it remained an open problem until Boneh *et al.* [19] proposed the first construction. The main idea of Boneh *et al.*'s scheme is to construct key-dependent encryptions without knowing the private key. When considering the case of KDM-CCA, unfortunately, Boneh *et al.*'s approach causes a direct attack: an adversary can construct an encryption of the private key, submit it to the decryption oracle and obtain the private key.

Supported by the National Basic Research Program of China (973 project, No.2014CB340603) and the National Nature Science Foundation of China (No.61379137, No.61272534).

Since the plaintexts may depend on the private key, existing techniques to achieve IND-CCA2 (Indistinguishability security against adaptive chosen ciphertext attacks) security in the standard model can not be used to construct KDM-CCA secure schemes directly. Camenisch, Chandran and Shoup [22] modified the Naor-Yung double encryption paradigm [41], and showed that one can combine a KDM-CPA (Key-dependent message security against chosen plaintext attacks) secure scheme with an IND-CCA2 secure scheme, along with an appropriate non-interactive zero-knowledge (NIZK) proof, to obtain a KDM-CCA secure scheme.

To construct practical KDM-CCA secure public key encryption (PKE) schemes, a direct solution is to replace the generic inefficient NIZK proof system by the hash proof system in [25]. Unfortunately, when the adversary can get encryptions of the private key of the hash proof system, the entropy of the private key will be leaked completely. To solve this problem, Hofheinz [33] proposed a “twice encryption” construction, in which the algorithm of the hash proof system shares the same private key with the encryption algorithm and two random coins are used: one for encryption and the other for hash proof. To prevent the adversary from generating valid ciphertexts of key-dependent messages, Hofheinz [33] added an authentication tag, constructed by embedding the plaintext into an encrypted LAF (Lossy Algebraic Filter), to the ciphertext. It guarantees that, in order to place a valid key-dependent decryption query, the adversary would have to guess the whole private key.

Galindo *et al.* [27] proposed a master key-dependent message (MKDM) secure IBE (Identity Based Encryption) scheme. Using the IBE to PKE transformation of Canetti, Halevi and Katz [24], they get a KDM-CCA secure PKE scheme. However, their concrete construction only achieves a bounded version of KDM security, that is, the adversary can only make a bounded number of encryption queries per public key.

1.1 Our Contribution

We propose an efficient KDM-CCA secure public key encryption scheme with respect to affine functions by enhancing an IND-CCA2 secure hybrid encryption scheme based on the high entropy hash proof system which was proposed by Kiltz *et al.* in [37]. Briefly, Kiltz *et al.* [37] provided a transformation from a k -entropic to a universal₂ hash proof system. Combining the latter with an AE-OT (Semantic and integrity security against one-time attack) secure data encapsulation mechanism (DEM) gives an IND-CCA2 secure hybrid encryption scheme. However, when key-dependent messages are encrypted by this hybrid encryption, the entropy of the private key of the hash proof system may be leaked completely.

Specifically, let $(u, e = \text{DEM.E}_k(m))$ be a ciphertext of such a hybrid encryption scheme, where $(u, k) = \text{KEM.E}_{pk}(r)$, KEM is the key encapsulation mechanism (KEM) constructed based on the hash proof system, pk is the public key, k is the encapsulated key. When key-dependent messages are encrypted by this hybrid encryption, for example $e = \text{DEM.E}_k(f(sk))$ (sk denotes the private key

of the KEM, $f(\cdot)$ denotes the key-dependent function), the entropy of the private key of the hash proof system may be leaked completely. To achieve KDM-CCA security, we enhance the hybrid encryption as follows:

- **Key-Dependent Message Encoding.** To deal with the problem of entropy leakage of key-dependent encryptions, we enhance the hybrid encryption by using a key-dependent message encoding $\mathcal{E}(\cdot)$. Specifically, instead of direct encryption, the plaintexts are encoded as $\mathcal{E}(m)$ before encryption. We require that, $\mathcal{E}(f(sk))$ can be constructed publicly without using the private key sk . Hence $\text{DEM.E}_k(\mathcal{E}(f(sk)))$ will not leak any entropy of sk . In fact most of the KDM-CPA secure public key encryption schemes [4, 19, 20, 40] meet this requirement.
- **Entropy Filter.** Although the key-dependent message encoding prevents the challenge ciphertexts from releasing the entropy of the private key of the hash proof system, it enables the adversary to construct valid key-dependent encryptions at the same time. Inspired by the technique of Hofheinz [33], we solve this problem by adding an authentication tag to the ciphertext. Specifically, we divide the entropy of the private key of the hash proof system into two parts, and use an entropy filter to derive the first part to construct an authentication tag. Let $\theta(\cdot)$ be an entropy filter, the DEM part of the hybrid encryption is enhanced as $\text{DEM.E}_k(\theta(m), \mathcal{E}(m))$.
- **RKA Secure Authenticated Encryption.** To guarantee that authentication tag $\theta(m)$ can be used to prevent the adversary from constructing valid ciphertexts, we must prevent the challenge ciphertexts from releasing the entropy of the private key of the hash proof system derived by the entropy filter. The main difficulty is that, to prevent the entropy leakage, the challenger needs to provide a random encapsulated key for each key-dependent encryption query. However, the entropy of the second part of the hash proof system is not enough to protect the encapsulated keys for all of the key-dependent encryption queries. We solve this problem by using an RKA secure authenticated encryption. Specifically, let k^* be an original key for the RKA secure authenticated encryption scheme, in the construction of challenge ciphertexts, the keys for the authenticated encryption are affine functions of k^* . And k^* is hidden from the adversary perfectly by using linearly dependent combinations of the second part of the private key. According to the definition of RKA security, the encryption scheme is secure if k^* is randomly distributed. Therefor, we can hide the authentication tag from the adversary perfectly.

On RKA Secure Authenticated Encryption. Related-key attacks (RKAs) were first proposed in [14, 38] as a cryptanalysis tool for block ciphers. Motivated by real attacks [15, 18], theoretical model for RKA was proposed by Bellare and Kohno [12]. In the last decade the RKA security for a wide range of cryptographic primitives was studied [1, 5, 9, 10, 13, 29, 30, 35, 36, 39, 43].

Up to now the RKA security for authenticated encryption has not been studied yet. We propose formal definition for the semantic security and integrity

security of authenticated encryption under RKAs. Similar to [12], we consider RKA security with respect to a class of related-key deriving (RKD) functions F which specify the key-relations available to the adversary. Informally, we let the adversary apply chosen plaintext attacks and forgery attacks with respect to a set of keys k_1, \dots, k_l which are derived from the original key k via a known function $f \in F$.

According to the framework in [10], it is easy to transform an AE-OT secure authenticated encryption scheme to an RKA secure authenticated encryption scheme based a RKA secure PRF (Pseudorandom function). Although the recent construction of RKA secure PRF with respect to affine function in [1] seems very efficient in general case, it will be very inefficient for our application. Specifically, the RKA secure PRF proposed by Abdalla *et al.* in [1] can be described as follows:

$$F(\mathbf{a}, x) = \text{NR}^*(\mathbf{a}, 11 || h(x, (g^{\mathbf{a}[1]}, \dots, g^{\mathbf{a}[n]}))), \text{NR}^*(\mathbf{a}, x) = g^{\prod_{i=1}^n \mathbf{a}[i]^{x[i]}},$$

where $\mathbf{a} = (\mathbf{a}[1], \dots, \mathbf{a}[n]) \in (Z_q^*)^{n+1}$ is the key for the PRF, $x = (x[1], \dots, x[n] \in \{0, 1\}^n)$ is the input of the PRF, n is the parameter of security level, $G = \langle g \rangle$ is a group of order q , h is a collision resistant hash function. When $(g^{\mathbf{a}[1]}, \dots, g^{\mathbf{a}[n]})$ are precomputed, the computation of F only needs one exponentiation. However, when embedded into our scheme, the key of the PRF \mathbf{a} is randomly generated for every ciphertext. That is, we need to compute $(g^{\mathbf{a}[1]}, \dots, g^{\mathbf{a}[n]})$ for every computation of the PRF. As a result, the computation of the PRF needs $n + 1$ exponentiations.

Moreover, the key space of the RKA secure PRF is a vector that contains n elements. To embed this PRF into our scheme, the KEM part of our scheme needs to encapsulate the key \mathbf{a} which contains n elements. This will significantly enlarge the ciphertext.

In this paper, we propose a direct construction of RKA secure authenticated encryption scheme with respect to affine functions that do not contain constant functions. Concretely, let $f(x) = ax + b$ be an affine function, we consider affine functions that $a \neq 0$. Let π be an AE-OT secure authenticated encryption scheme, G a group with order N , $g \in G$ a generator of G , $H : Z_N \rightarrow \{0, 1\}^{l_\kappa}$ a 4-wise independent hash function, r a random number chosen from Z_N , our RKA secure authenticated encryption scheme encrypts the plaintext message m as follows:

$$u \leftarrow g^r, \kappa \leftarrow H(u^k, u), e \leftarrow \pi.E_\kappa(m),$$

where l_κ is the length of κ , $\pi.E_\kappa(\cdot)$ denote the encryption algorithm with the key κ , and the ciphertext is (u, e) . We prove that if π is AE-OT secure and the DDH (Decisional Diffie-Hellman) assumption holds in G , then our new authenticated encryption scheme is RKA secure.

Technical Relation to Hofheinz's Scheme. To prevent the entropy leakage of the authentication tag added to the ciphertext, Hofheinz's [33] solution is embedding the plaintext into an encrypted LAF. Concretely, for a given public key Fpk , if t is a lossy tag, then $\text{LAF}_{Fpk, t}([sk]_{Z_p^n})$ only depends on a linear

combination of its input $[sk]_{Z_p^n} \in Z_p^n$, here $[sk]_{Z_p^n}$ denotes encoding the private key sk into Z_p^n . In particular, the coefficients of the linear combination only depend on Fpk . That is, for different tags t_i , $\text{LAF}_{Fpk, t_i}([sk]_{Z_p^n})$ only leaks the same linear function of $[sk]_{Z_p^n}$. However, when considering KDM security with respect to richer functions such as affine functions, $\text{LAF}_{Fpk, t_i}([f_i(sk)]_{Z_p^n})$ may leak all the entropy of $[sk]_{Z_p^n}$, here f_1, \dots, f_l are affine functions chosen by the adversary. Thus, Hofheinz's scheme can only achieve CIRC-CCA (circular security against chosen ciphertext attacks) security.

In our new construction, different from the approach of Hofheinz [33], we divide the entropy of the private key into two independent parts and derive the first part by using an entropy filter to construct an authentication tag. We prove that the entropy in the first part of the private key is enough to prevent the adversary from constructing a valid key-dependent encryption. Compared with the LAF used by Hofheinz, the construction of our entropy filter is simpler and more efficient.

To prevent the entropy leakage of the authentication tag in the challenge ciphertext, Hofheinz's [33] solution is “lossy”. The lossy property of LAF guarantees that, information theoretically, little information about the private key is released. To this end, we use a way of “hiding”. Concretely, the keys for the authenticated encryption are hidden from the adversary by using linear combinations of the second part of the private key. According to the definition of RKA security, the encryption scheme is secure even when the keys are linearly dependent. Thus, the authentication tags are perfectly hidden from the adversary.

1.2 Outline

In section 2 we review the definitions of public key encryption scheme, KDM-CCA security, decisional composite residuosity assumption, authenticated encryption, decisional Diffie-Hellman assumption and leftover hash lemma. In section 3 we propose the formal definition of RKA secure authenticated encryption scheme and an efficient construction. In section 4 we propose our new KDM-CCA secure scheme and the security proof. Finally we give the conclusion in section 5.

2 Definitions

We write $[n] = \{1, \dots, n\}$. In describing probabilistic processes, if S is a finite set, we write $s \xleftarrow{R} S$ to denote assignment to s of an element sampled from uniform distribution on S . If A is a probabilistic algorithm and x an input, then $A(x)$ denotes the output distribution of A on input x . Thus, we write $y \leftarrow A(x)$ to denote of running algorithm A on input x and assigning the output to the variable y . For an integer $v \in N$, we let U_v denote the uniform distribution over $\{0, 1\}^v$, the bit-string of length v . The *min-entropy* of a random variable X is defined as

$$H_\infty(X) = -\lg(\max_{x \in \mathcal{X}} \Pr[X = x]).$$

The statistical distance between two random variables X, Y is defined by

$$\text{SD}(X, Y) = \frac{1}{2} \sum_x |\Pr[X = x] - \Pr[Y = x]|.$$

2.1 Public Key Encryption Scheme

A public key encryption scheme consists of the following algorithms:

- **Setup(l)**: A probabilistic polynomial-time setup algorithm takes as input a security parameter l and outputs the system parameter prm . We write $prm \leftarrow \text{Setup}(l)$.
- **Gen(prm)**: A probabilistic polynomial-time key generation algorithm takes as input the system parameter prm and outputs a public key pk and a private key sk . We write $(pk, sk) \leftarrow \text{Gen}(prm)$.
- **E(pk, m)**: A probabilistic polynomial-time encryption algorithm takes as input a public key pk and a message m , and outputs a ciphertext c . We write $c \leftarrow \text{E}_{pk}(m)$.
- **D(sk, c)**: A decryption algorithm takes as input a ciphertext c and a private key sk , and outputs a plaintext m . We write $m \leftarrow \text{D}_{sk}(c)$.

For correctness, we require $\text{D}_{sk}(\text{E}_{pk}(m)) = m$ for all (pk, sk) output by $\text{Gen}(prm)$ and all $m \in M$ (M denotes the message space).

2.2 KDM-CCA Security

A public key encryption scheme is n -KDM-CCA secure w.r.t F if the advantage of any adversary in the following game is negligible in the security parameter l :

- **Step 1:** The challenger runs $\text{Setup}(l)$ to generate the system parameter prm , then runs $\text{Gen}(prm)$ to obtain n keypairs $(pk_i, sk_i), i = 1, \dots, n$. It sends prm and (pk_1, \dots, pk_n) to the adversary.
- **Step 2:** The adversary issues decryption queries with (c_j, i) , where $1 \leq j \leq Q_d, 1 \leq i \leq n$, Q_d denotes the total number of decryption queries. With each query, the challenger sends $m_j \leftarrow \text{D}_{sk_i}(c_j)$ to the adversary.
- **Step 3:** The adversary issues encryption queries with (f_λ, i) , where $f_\lambda \in F, 1 \leq \lambda \leq Q_e, 1 \leq i \leq n$, Q_e denotes the total number of encryption queries. With each query, the challenger sends $c_\lambda^* \leftarrow \text{E}_{pk_i}(m_{\lambda b})$ to the adversary, where $m_{\lambda 0} = \{0\}^{l_\lambda}, m_{\lambda 1} = f_\lambda(sk_1, \dots, sk_n), l_\lambda = |m_{\lambda 1}|, b \in \{0, 1\}$ is a random bit selected by the challenger (note that the challenger chooses b only once).
- **Step 4:** The adversary issues decryption queries just as in step 2, the only restriction is that the adversary can not ask the decryption of c_λ^* for $1 \leq \lambda \leq Q_e$.
- **Step 5:** Finally, the adversary outputs b' as the guess of b .

The adversary's advantage is defined as:

$$\text{Adv}_{\mathcal{A},n}^{\text{cca}} = |\Pr[b' = b] - 1/2|.$$

As a special case of n -KDM-CCA, if $F = \{f_\lambda : f_\lambda(sk_1, \dots, sk_n) = sk_\lambda, \lambda \in [n]\}$ we say that the public key encryption scheme is n -CIRC-CCA secure.

2.3 Decisional Composite Residuosity Assumption

Let $N = pq$, p and q are safe primes, the quadratic residuosity group over $Z_{N^s}^*$ is defined as $\text{QR}_{N^s} = \{u^2 \pmod{N^s} | u \in Z_{N^s}^*\}$, the square composite residuosity group as $\text{SCR}_{N^s} = \{v^{N^{s-1}} \pmod{N^s} | v \in \text{QR}_{N^s}\}$, the root of the unity group as $\text{RU}_{N^s} = \{T^r \pmod{N^s} | r \in [N^{s-1}], T = 1 + N \pmod{N^s}\}$, consider the experiment $\text{Exp}_{\mathcal{A}}^{\text{dcr}}$:

$$W_0 \xleftarrow{R} \text{QR}_{N^s}, W_1 \xleftarrow{R} \text{SCR}_{N^s}, b \xleftarrow{R} \{0, 1\},$$

$$b' \leftarrow \mathcal{A}(N, W_b), \text{return } b'.$$

Denote $\Pr[\text{Suc}_{\mathcal{A}}^{\text{dcr}}] = \Pr[b' = b]$ as the probability that \mathcal{A} succeeds in guessing b . We define the advantage of \mathcal{A} in $\text{Exp}_{\mathcal{A}}^{\text{dcr}}$ as

$$\text{Adv}_{\mathcal{A}}^{\text{dcr}} = |\Pr[b' = b] - \frac{1}{2}|.$$

We say that the decisional composite residuosity (DCR) assumption holds if $\text{Adv}_{\mathcal{A}}^{\text{dcr}}$ is negligible for all polynomial-time adversaries \mathcal{A} .

We review a lemma of [20] which is useful to the security proof of our scheme. Let \mathcal{A} be an adversary and $s \geq 2$ be an integer, define game IV2 as follows.

$$\text{IV2: } g_1, g_2 \xleftarrow{R} \text{SCR}_{N^s}, b' \leftarrow \mathcal{A}^{\mathcal{O}_{\text{iv2}}(\delta, \bar{\delta})}(N, g_1, g_2).$$

In the game above \mathcal{A} is allowed to make polynomial number queries. In each query, \mathcal{A} can send $(\delta, \bar{\delta} \in Z_{N^{s-1}})$ to the oracle \mathcal{O}_{iv2} . $\mathcal{O}_{\text{iv2}}(\delta, \bar{\delta})$ then selects $r \in [\lfloor N/4 \rfloor]$ randomly and returns $(g_1^r T^\delta, g_2^r T^{\bar{\delta}})$ if $b = 1$ and (g_1^r, g_2^r) otherwise, where $b \in \{0, 1\}$ is randomly selected. The advantage of \mathcal{A} is defined to be

$$\text{Adv}_{\mathcal{A}}^{\text{iv2}} = |\Pr[b' = b] - \frac{1}{2}|.$$

Lemma 1. *No polynomial-time adversary can have non-negligible advantage in IV2 under the DCR assumption.*

Our definition of IV2 follows from the version in [40], which is slightly different from the original definition in [20].

Note that the discrete logarithm $\text{dlog}_T(X) := x$ for $X \in \text{RU}_{N^s}$ and $x \in N^{s-1}$ can be efficiently computed [26, 42].

2.4 Authenticated Encryption

We review the security definitions of the authenticated encryption scheme. An authenticated encryption (AE) scheme consists of three algorithms:

- AE.Setup(l): The setup algorithm takes as input the security parameter l , and outputs public parameters $param$ and the key k of the AE scheme. We write $(param, k) \leftarrow \text{AE}.\text{Setup}(l)$.
- AE.E _{k} (m): The encryption algorithm takes as inputs a key k and a message m and outputs a ciphertext χ . We write $\chi \leftarrow \text{AE}.\text{E}_k(m)$
- AE.D _{k} (χ): The decryption algorithm takes as inputs a key k , a ciphertext χ and outputs a message m or the rejection symbol \perp . We write $m \leftarrow \text{AE}.\text{D}_k(\chi)$.

We require that for all $k \in \{0, 1\}^{l_k}$ (l_k denotes the length of k), $m \in \{0, 1\}^*$, we have:

$$\text{AE}.\text{D}_k(\text{AE}.\text{E}_k(m)) = m.$$

An AE scheme is IND-OT (indistinguishability against one-time attacks) secure if the advantage of any PPT (Probabilistic Polynomial Time) adversary \mathcal{A} in the following game is negligible in the security parameter l :

1. The challenger randomly generates an appropriately sized key k .
2. The adversary \mathcal{A} queries the encryption oracle with two messages m_0 and m_1 such that $|m_0| = |m_1|$. The challenger computes

$$b \xleftarrow{R} \{0, 1\}, \chi^* \leftarrow \text{AE}.\text{E}_k(m_b)$$

and responds with χ^* .

3. Finally, \mathcal{A} outputs a guess b' .

The advantage of \mathcal{A} is defined as $\text{Adv}_{\mathcal{A}}^{\text{ind-ot}}(l) = |\Pr[b = b'] - 1/2|$. We say that the AE is one-time secure in the sense of indistinguishability if $\text{Adv}_{\mathcal{A}}^{\text{ind-ot}}(l)$ is negligible.

An AE scheme is INT-OT (one-time secure in the sense of ciphertext integrity) secure if the advantage of any PPT adversary \mathcal{A} in the following game is negligible in the security parameter l :

1. The challenger randomly generates an appropriately sized key k .
2. The adversary \mathcal{A} queries the encryption oracle with a message m . The challenger computes

$$\chi^* \leftarrow \text{AE}.\text{E}_k(m)$$

and responds with χ^* .

3. Finally, the adversary \mathcal{A} outputs a ciphertext $\chi \neq \chi^*$ such that $\text{AE}.\text{D}_k(\chi) \neq \perp$.

The advantage of \mathcal{A} is defined as

$$\text{Adv}_{\mathcal{A}}^{\text{int-ot}}(l) = \Pr[\text{AE.D}_k(\chi) \neq \perp].$$

We say that the AE is one-time secure in the sense of ciphertext integrity if $\text{Adv}_{\mathcal{A}}^{\text{int-ot}}(l)$ is negligible. An AE is one-time secure (AE-OT) iff it is IND-OT secure and INT-OT secure.

2.5 Decisional Diffie-Hellman Assumption

Let G be a group of large prime order q , g is a generator of G , consider the experiment $\text{Exp}_{G,\mathcal{A}}^{\text{ddh}}$:

$$(x, y, z) \xleftarrow{R} Z_q^*; W_0 \leftarrow g^z; W_1 \leftarrow g^{xy}; b \xleftarrow{R} \{0, 1\}$$

$$b' \leftarrow \mathcal{A}(g, g^x, g^y, W_b).$$

We define the advantage of \mathcal{A} as

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

We say that the DDH assumption holds if $\text{Adv}_{\mathcal{A}}^{\text{ddh}}$ is negligible for all polynomial-time adversaries \mathcal{A} .

2.6 Leftover Hash Lemma

Multiple versions of LHL (Leftover Hash Lemma) have been proposed, we recall the generalized version in [37]: if H is 4-wise independent, then $(H, H(X), H(\tilde{X}))$ is close to uniformly random distribution.

Lemma 2. (Generalized Leftover Hash Lemma) *Let $\mathcal{H} = \{H : \mathcal{X} \rightarrow \{0, 1\}^v\}$ be a family of 4-wise independent hash functions, $(X, \tilde{X}) \in \mathcal{X} \times \mathcal{X}$ be two random variables where $H_\infty(X) \geq \kappa$, $H_\infty(\tilde{X}) \geq \kappa$ and $\Pr[X = \tilde{X}] \leq \delta$. Then for $H \xleftarrow{R} \mathcal{H}$ and $U_{2l} \xleftarrow{R} \{0, 1\}^{2l}$,*

$$SD((H, H(X), H(\tilde{X})), (H, U_{2l})) \leq \sqrt{1 + \delta} \cdot 2^{l - \kappa/2} + \delta.$$

3 RKA Secure Authenticated Encryption

3.1 Definition of RKA Security for Authenticated Encryption

Following [12], we give a formal definition for the RKA security of AE schemes. Similar as the definition of AE-OT security, the definition of RKA security includes two aspects: indistinguishability security against related key attacks (IND-RKA) and integrity security against related key attacks (INT-RKA).

Definition 1. An AE scheme is IND-RKA secure with respect to a class of related-key deriving functions F if the advantage of any PPT adversary \mathcal{A} in the following game is negligible in the security parameter l .

1. The challenger randomly generates an appropriately sized key k for the security parameter l .
2. The adversary \mathcal{A} makes a sequence of related-key encryption queries with m_{i0}, m_{i1}, f_i such that $|m_{i0}| = |m_{i1}|$. Here $1 \leq i \leq Q$, Q denotes the number of related-key encryption queries made by the adversary \mathcal{A} , m_{i0} and m_{i1} are two messages, $f_i \in F$, F is a class of related-key deriving functions. The challenger chooses $b \in \{0, 1\}$ randomly and responds with a challenge ciphertext for each query of \mathcal{A} computed as follows:

$$\chi_i^* \leftarrow \text{AE.E}_{f_i(k)}(m_{ib}).$$

3. Finally, \mathcal{A} outputs a guess b' .

The advantage of \mathcal{A} is defined as $\text{Adv}_{\mathcal{A}}^{\text{ind-rka}}(l) = |\Pr[b = b'] - 1/2|$.

Definition 2. An AE scheme is INT-RKA secure with respect to a class of related-key deriving functions F , if the advantage of any PPT adversary \mathcal{A} in the following game is negligible in the security parameter l .

1. The challenger randomly generates an appropriately sized key k for the security parameter l .
2. The adversary \mathcal{A} makes a sequence of related-key encryption queries with m_i, f_i . Here $1 \leq i \leq Q$, Q denotes the number of related-key encryption queries made by the adversary \mathcal{A} , m_i is a plaintext message, $f_i \in F$, F is a class of related-key deriving functions. The challenger responds with a challenge ciphertext for each query of \mathcal{A} computed as follows:

$$\chi_i^* \leftarrow \text{AE.E}_{f_i(k)}(m_i).$$

3. Finally, the adversary \mathcal{A} outputs a ciphertext χ and a related-key deriving function f such that $(f, \chi) \neq (f_i, \chi_i^*)$ and $\text{AE.D}_{f(k)}(\chi) \neq \perp$.

The advantage of \mathcal{A} is defined as

$$\text{Adv}_{\mathcal{A}}^{\text{int-rka}}(l) = \Pr[\text{AE.D}_k(\chi) \neq \perp].$$

Finally, the RKA security of the AE scheme is defined as follows:

Definition 3. An AE is RKA secure iff it is IND-RKA secure and INT-RKA secure.

3.2 Construction of RKA Secure Authenticated Encryption

We propose a randomized RKA secure authenticated encryption scheme with respect to affine functions that do not contain constant functions. Concretely, let $f(x) = ax + b$ be an affine function, we consider affine functions that $a \neq 0$.

Let π be an AE-OT secure authenticated encryption scheme, our new construction $\bar{\pi}$ is described as follows:

- **Setup(l):** Randomly choose two safe primes p and q that $2pq + 1$ is also a prime, then compute:

$$N \leftarrow pq, \bar{N} \leftarrow 2N + 1, g \xleftarrow{R} \text{QR}_{\bar{N}}, \text{param} \leftarrow (N, \bar{N}, g), k \xleftarrow{R} Z_N.$$

- **Encryption:** The encryption algorithm takes as inputs a key $k \in Z_N$ and a message m and computes as follows:

$$r \xleftarrow{R} Z_N, u \leftarrow g^r, \kappa \leftarrow \text{H}(u^k, u),$$

$$e \leftarrow \pi.\text{E}_\kappa(m), \chi \leftarrow (u, e).$$

Here $\text{H} : \text{QR}_{\bar{N}} \times \text{QR}_{\bar{N}} \rightarrow \{0, 1\}^{l_\kappa}$ is a 4-wise independent universal hash function, l_κ is the length of κ .

- **Decryption:** The decryption algorithm takes as inputs a key k , a ciphertext $\chi = (u, e)$ and computes as follows:

$$\kappa \leftarrow \text{H}(u^k, u), m \leftarrow \pi.\text{D}_\kappa(e).$$

We prove that if π is AE-OT secure and the DDH assumption holds in $\text{QR}_{\bar{N}}$, then our new authenticated encryption scheme is RKA secure.

Theorem 1. *Assume the DDH assumption holds in $\text{QR}_{\bar{N}}$, π is AE-OT secure, then $\bar{\pi}$ is RKA secure with respect to affine functions $f(x) = ax + b$ that $a \neq 0$.*

According to the definition of the RKA security of AE scheme, we need to prove two lemmas as follows.

Lemma 3. *Assume the DDH assumption holds in $\text{QR}_{\bar{N}}$, π is AE-OT secure, then $\bar{\pi}$ is IND-RKA secure with respect to affine functions $f(x) = ax + b$ that $a \neq 0$.*

Lemma 4. *Assume the DDH assumption holds in $\text{QR}_{\bar{N}}$, π is AE-OT secure, then $\bar{\pi}$ is INT-RKA secure with respect to affine functions $f(x) = ax + b$ that $a \neq 0$.*

Proof of Lemma 3: The proof is via a sequence of games involving the challenger \mathcal{C} and the adversary \mathcal{A} . Let W_i be the event that \mathcal{A} guesses b correctly in Game i .

- **Game 0:** This game is the actual IND-RKA game. Hence, we have:

$$\Pr[W_0] = 1/2 + \text{Adv}_{\mathcal{A}}^{\text{ind-rka}}. \quad (1)$$

- **Game 1:** This game is exactly like Game 0, except that the challenge ciphertexts are computed using g^k instead of k . That is, let $\hat{g} = g^k$, the keys for π are computed as $\kappa_i \leftarrow H(\hat{g}^{r_i a_i} g^{r_i b_i}, g^{r_i})$. It is clear that Game 0 and Game 1 are identical from the point view of the adversary \mathcal{A} , hence we have:

$$\Pr[W_0] = \Pr[W_1]. \quad (2)$$

- **Game 1.i:** Denote Game 1.0 as Game 1, for $1 \leq i \leq Q$, Game 1.i is exactly like Game 1.($i-1$) except that the key for π in the i th ciphertext is computed as $\kappa_i \leftarrow H(g^{r_i^* a_i} g^{r_i b_i}, g^{r_i})$, where $r_i^* \neq r_i$ is randomly chosen from Z_N . It is clear that if the adversary \mathcal{A} can distinguish Game 1.i from Game 1.($i-1$), then we can break the DDH assumption. Briefly, given a DDH challenge $(g, \hat{g}, g^r, \hat{g}^{r^*})$, to compute the i th ciphertext, the challenger sets $u_i = g^r, \kappa_i = H(g^{r^* a_i} g^{r b_i}, u_i)$. The computation of the other ciphertexts is the same as in Game 1.($i-1$). It is clear that when $r = r^*$, it is just the case in Game 1.($i-1$), when $r \neq r^*$ it is just the case in Game 1.i. Thus, if the adversary \mathcal{A} can distinguish Game 1.i from Game 1.($i-1$), then the challenger can break the DDH assumption. Hence we have:

$$\Pr[W_{1.i-1}] \leq \Pr[W_{1.i}] + \text{Adv}_{\mathcal{A}}^{\text{ddh}}. \quad (3)$$

- **Game 2:** This game is exactly Game 1.Q. It is clear that, when $a_i \neq 0$ the keys for the challenge ciphertexts $\kappa_i = H(\hat{g}^{r_i^* a_i} g^{r_i b_i}, g^{r_i})$ are randomly distributed. According to the IND-OT security of π , we have:

$$\Pr[W_2] \leq 1/2 + Q\text{Adv}_{\mathcal{A}}^{\text{ind-ot}}. \quad (4)$$

From equations (1) – (4) we have:

$$\text{Adv}_{\mathcal{A}}^{\text{ind-rka}} \leq Q\text{Adv}_{\mathcal{A}}^{\text{ddh}} + Q\text{Adv}_{\mathcal{A}}^{\text{ind-ot}}.$$

This completes the proof of lemma 3. \square

Proof of Lemma 4: The proof is via a sequence of games involving the challenger \mathcal{C} and the adversary \mathcal{A} . Let W_i be the event that \mathcal{A} outputs a valid ciphertext in Game i .

- **Game 0:** This game is the actual INT-RKA game. When responding to a related-key encryption query with m_i, f_i , where $f_i(k) = a_i k + b_i$, \mathcal{C} computes as follows:

$$\begin{aligned} r_i &\xleftarrow{R} Z_N, u_i \leftarrow g^{r_i}, \kappa_i \leftarrow H(u_i^{a_i k + b_i}, u_i), \\ e_i &\leftarrow \pi.E_{\kappa_i}(m_i), \chi_i \leftarrow (u_i, e_i). \end{aligned}$$

- **Game 1:** This game is exactly like Game 0, except that the challenge ciphertexts are computed using g^k instead of k . That is, let $\hat{g} = g^k$, the keys for π are computed as $\kappa_i \leftarrow H(\hat{g}^{r_i a_i} g^{r_i b_i}, g^{r_i})$. It is clear that Game 0 and Game 1 are identical from the point view of the adversary \mathcal{A} , hence we have:

$$\Pr[W_0] = \Pr[W_1]. \quad (5)$$

- **Game 1.i:** Denote Game 1.0 as Game 1, for $1 \leq i \leq Q$, Game 1.i is exactly like Game 1.(i-1) except that the key for π in the i th ciphertext is computed as $\kappa_i \leftarrow H(\hat{g}^{r_i^* a_i} g^{r_i b_i}, g^{r_i})$, where $r_i^* \neq r_i$ is randomly chosen from Z_N . It is clear that if the adversary \mathcal{A} can distinguish Game 1.i from Game 1.(i-1), then we can break the DDH assumption. Hence we have:

$$\Pr[W_{1,i-1}] \leq \Pr[W_{1,i}] + \text{Adv}_{\mathcal{A}}^{\text{ddh}}. \quad (6)$$

- **Game 2:** This game is exactly Game 1.Q. Denote $\chi = (u = g^r, e = \pi.E_\kappa(m), f(k) = ak + b)$ as the forged ciphertext and the related-key deriving function, we consider three cases as follows:

- Case 1: $f = f_i, u = u_i, e \neq e_i$. In this case we have $\kappa = \kappa_i$, if χ is a valid ciphertext then we get a forged ciphertext of e_i . It is clear that the keys for the challenge ciphertexts $\kappa_i = H(\hat{g}^{r_i^* a_i} g^{r_i b_i}, g^{r_i})$ are randomly distributed. According to the INT-OT security of π , we have:

$$\Pr[W_2 | \text{Case1}] \leq Q \text{Adv}_{\mathcal{A}}^{\text{int-ot}}. \quad (7)$$

- Case 2: $f \neq f_i, u = u_i$. In this case, we consider two subcases as follows:
 - * Case 2.1: $r_i^* ak + r_i b = r_i^* a_i k + r_i b_i$. It is clear that, if the adversary \mathcal{A} submits such related-key deriving function, we can break the discrete logarithm assumption. Concretely, given $g, \hat{g} = g^k$, the simulator \mathcal{S} can play Game 2 with \mathcal{A} . When \mathcal{A} submits χ, f , \mathcal{S} computes $k_i = \frac{r_i(b_i - b)}{r_i^*(a - a_i)}$ and test whether $\hat{g} = g^{k_i}$. Let ϵ_{dlg} be the probability that any adversary breaks the discrete logarithm assumption, we have:

$$\Pr[\text{Case2.1}] \leq \epsilon_{\text{dlg}}. \quad (8)$$

- * Case 2.2: $r_i^* ak + r_i b \neq r_i^* a_i k + r_i b_i$. In this subcase, $\hat{g}^{r_i^* a} g^{r_i b} \neq \hat{g}^{r_i^* a_i} g^{r_i b_i}$. Since r_i^* is randomly selected from Z_N and $a \neq 0, a_i \neq 0$, we have:

$$H_\infty(\hat{g}^{r_i^* a} g^{r_i b}) = H_\infty(\hat{g}^{r_i^* a_i} g^{r_i b_i}) = l_N,$$

where l_N is the length of N . According to the property of 4-wise independent hash functions, $H(\hat{g}^{r_i^* a} g^{r_i b}, u), H(\hat{g}^{r_i^* a_i} g^{r_i b_i}, u_i)$ is randomly distributed from the point view of the adversary \mathcal{A} . Hence, we have:

$$\Pr[W_2 | \text{Case2.2}] \leq \text{Adv}_{\mathcal{A}}^{\text{int-ot}}. \quad (9)$$

- Case 3: $u \neq u_i$. Since the information of k is statistically hidden by r_i^* , we have:

$$H_\infty(u^{ak+b}) = H_\infty(u_i^{a_i k + b_i}) = l_N,$$

where l_N is the length of N . According to the property of 4-wise independent hash functions, $H(u^{ak+b}, u), H(u_i^{a_i k + b_i}, u_i)$ is randomly distributed from the point view of the adversary \mathcal{A} . Hence, we have:

$$\Pr[W_2 | \text{Case3}] \leq \text{Adv}_{\mathcal{A}}^{\text{int-ot}}. \quad (10)$$

From equations (7) – (10) we have:

$$\Pr[W_2] \leq Q\text{Adv}_{\mathcal{A}}^{\text{int-ot}} + \epsilon_{\text{dlg}}. \quad (11)$$

From the equations (5), (6), (11) we have that:

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{int-rka}} &= \Pr[W_0] \\ &\leq Q\text{Adv}_{\mathcal{A}}^{\text{ddh}} + Q\text{Adv}_{\mathcal{A}}^{\text{int-ot}} + \epsilon_{\text{dlg}}. \end{aligned} \quad (12)$$

This completes the proof of lemma 4. \square

4 KDM-CCA Secure Scheme

4.1 The Idea of Our Construction

Before formal description, we give some intuition of our construction. Just as we mentioned in section 1.1, existing KDM-CPA schemes [4, 19, 20, 40] are very suitable to be used as a key-dependent message encoding. To encode the messages depending on the private key of the KEM part, the encoding scheme must share the same parameters (especially the private key) with the KEM part. For this reason, we first choose a KDM-CPA scheme, then translate it into a KEM scheme. Concretely, we choose the KDM-CPA scheme in [40] as the key-dependent message encoding scheme and translate it into a KEM by replacing the plaintext with a randomly selected key k . Briefly the ciphertext of the hybrid encryption can be described as:

$$u = g^r, e = h^r T^k, v = \bar{\pi} \cdot \text{E}_k(g^{\tilde{r}} || h^{\tilde{r}} T^m),$$

where $g \in \text{SCR}_{N^s}, s \geq 2, x \in [|N^2/4|], h = g^x, r, \tilde{r} \in [|N/4|]$, $\bar{\pi}$ is an RKA secure authenticated encryption scheme. Given the public key (g, h) , we have $h = g^x \pmod{\phi(N)/4} \pmod{N^s}$. That is, conditioned on the public key, $x \pmod{N}$ is hidden from the adversary information theoretically. Thus, the entropy of the private key is $x \pmod{N}$.

To divide the entropy of the private key into two parts independent of each other, we enlarge the entropy by extending the public key and the private key. As a result, the ciphertext of the hybrid encryption is extended as:

$$u_1 = g_1^r, u_2 = g_2^r, e = h^r T^k, v = \bar{\pi} \cdot \text{E}_k(g_1^{\tilde{r}} || g_2^{\tilde{r}} || h^{\tilde{r}} T^m),$$

where $g_1, g_2 \in \text{SCR}_{N^s}, x_1, x_2 \in [|N^2/4|], h = g_1^{x_1} g_2^{x_2}$. It is clear that, conditioned on the public key, the adversary can only get

$$h = g_1^{x_1} \pmod{\phi(N)/4} g_2^{x_2} \pmod{\phi(N)/4} \pmod{N^s}.$$

The entropy of the private key can be divided into two parts: the first part is the entropy contained in $(x_1 \pmod{\phi(N)/4}, x_2 \pmod{\phi(N)/4})$, the second part is the entropy contained in $(x_1 \pmod{N}, x_2 \pmod{N})$. It is easy to construct an

entropy filter to derive the first part of the entropy from the key-dependent messages as $\theta(f(x_1, x_2)) = g_1^{f(x_1, x_2)}$. To prevent the adversary from getting a valid key-dependent encryption by modifying the challenge ciphertexts, we use the universal hash function H to construct the authentication tag as $t = H(u_1||u_2||e||\theta(f(x_1, x_2)))$.

In order to protect the key of the RKA secure authenticated encryption scheme by using the second part of the entropy, we modify the KEM part of the hybrid encryption slightly as:

$$u_1 = g_1^r \pmod{N^2}, u_2 = g_2^r \pmod{N^2}, e = h^r T^k \pmod{N^2}.$$

Since $g_1 \pmod{N^2}, g_2 \pmod{N^2} \in \text{SCR}_{N^2}$, the KEM part is now computed in the group of SCR_{N^2} . In this case we have $e = h^r T^k \pmod{N^2} = h^r T^k \pmod{N} \pmod{N^2}$. That is, the range of k is now shrunk to $[N]$ from $[N^{s-1}]$. When we use (x_1, x_2) to protect k , only the second part of the entropy will be derived out.

4.2 The Proposed Scheme

- **Setup(l):** Randomly choose two safe primes p and q that $2pq + 1$ is also a prime, then compute:

$$N \leftarrow pq, g_1, g_2 \xleftarrow{R} \text{SCR}_{N^s}, \text{prm} \leftarrow (N, g_1, g_2).$$

- **Key Generation:** For prm , the key generation algorithm computes:

$$x_1, x_2 \xleftarrow{R} [\lfloor N^2/4 \rfloor], h \leftarrow g_1^{-x_1} g_2^{-x_2},$$

$$pk \leftarrow (h), sk \leftarrow (x_1, x_2).$$

- **Encryption:** For $m \in [N^{s-1}]$, the encryption algorithm computes the ciphertext c as follows:

$$r, \tilde{r} \xleftarrow{R} [\lfloor N/4 \rfloor], k \xleftarrow{R} Z_N,$$

$$u_1 \leftarrow g_1^r \pmod{N^2}, u_2 \leftarrow g_2^r \pmod{N^2}, e \leftarrow h^r T^k \pmod{N^2},$$

$$\tilde{u}_1 \leftarrow g_1^{\tilde{r}} \pmod{N^s}, \tilde{u}_2 \leftarrow g_2^{\tilde{r}} \pmod{N^s}, \tilde{e} \leftarrow h^{\tilde{r}} T^m \pmod{N^s},$$

$$t \leftarrow H(u_1||u_2||e||(g_1^m \pmod{N})), v \leftarrow \bar{\pi}.\text{E}_k(t||\tilde{u}_1||\tilde{u}_2||\tilde{e}), c \leftarrow (u_1, u_2, e, v).$$

where $H : \{0, 1\}^* \rightarrow \{0, 1\}^{l_t}$ is a universal hash function.

- **Decryption:** If $eu_1^{x_1} u_2^{x_2} \notin RU_{N^2}$ return the rejection symbol \perp , else compute:

$$k \leftarrow \text{dlog}_T(eu_1^{x_1} u_2^{x_2}),$$

$$t||\tilde{u}_1||\tilde{u}_2||\tilde{e} \leftarrow \bar{\pi}.\text{D}_k(v),$$

If $\tilde{e}\tilde{u}_1^{x_1}\tilde{u}_2^{x_2} \notin RU_{N^s}$ return \perp ,

$$m \leftarrow \text{dlog}_T(\tilde{e}\tilde{u}_1^{x_1}\tilde{u}_2^{x_2}),$$

If $t = H(u_1||u_2||e||(g_1^m \pmod{N}))$ return m , else return \perp .

It is clear that compared with Hofheinz's CIRC-CCA scheme [33], our new scheme is simpler and more efficient. Concretely, the ciphertext of the Hofheinz's scheme contains 6 Z_{N^3} -elements, 43 Z_p -elements ($p \approx N/4$), a chameleon hash randomness, a one-time signature and the verification key, and a symmetric ciphertext (whose size is about one N_{N^2} -element plus some encryption randomness). However, to achieve CIRC-CCA security, we can set $s = 3$ for our scheme, in this case the ciphertext of our new scheme only contains 3 Z_{N^2} -elements, 3 Z_{N^3} -elements, 1 $Z_{\bar{N}}$ -elements ($\bar{N} = 2N + 1$), a hash value (whose size is about the security parameter l) and the ciphertext expansion of the authenticated encryption scheme (whose size is about twice of the security parameter l).

4.3 Security Proof

Before formal proof, we give an intuition of the 1-KDM-CCA security of our scheme. The n -KDM-CCA security can be achieved by re-randomizing keys and ciphertexts of a single instance of the scheme like in [19, 20]. Our main idea is to achieve KDM-CCA security based on the entropy of the private key. Concretely, let $x_1^{hide} = x_1 \bmod N$, $x_1^{real} = x_1 \bmod \phi(N)/4$, $x_2^{hide} = x_2 \bmod N$, $x_2^{real} = x_2 \bmod \phi(N)/4$, we have that $h = g_1^{-x_1} g_2^{-x_2} = g_1^{-x_1^{real}} g_2^{-x_2^{real}}$. Hence, x_1^{hide} and x_2^{hide} are information theoretically hidden from the adversary, conditioned on the public key.

In the security reduction, to encrypt $f(x_1, x_2) = a_1 x_1 + a_2 x_2 + b$, the ciphertext is constructed as $\bar{\pi}.E_k(t||g_1^{\bar{T}} T^{a_1}||g_2^{\bar{T}} T^{a_2}||h^{\bar{T}} T^b)$. In fact, this ciphertext can be constructed without knowing x_1^{hide} and x_2^{hide} . Thus it will not leak any information of x_1^{hide} and x_2^{hide} .

Let $\log_{g_1} g_2 = w$, we have that, conditioned on the public key, the only information that the adversary gets about x_1^{real} and x_2^{real} is the following equation:

$$\log_{g_1} h = -(x_1^{real} + wx_2^{real}) \bmod \phi(N)/4.$$

It is clear that there is entropy in the pair (x_1^{real}, x_2^{real}) , since the adversary only gets one equation for two variables. We divide the entropy of the private key into two parts: the first part is the entropy contained in (x_1^{real}, x_2^{real}) , the second part is the entropy contained in (x_1^{hide}, x_2^{hide}) .

The first part of the entropy is embedded into an authentication tag to prevent the adversary from constructing a valid encryption of an affine function of the private key $f(x_1, x_2) = a_1 x_1 + a_2 x_2 + b$. Concretely, we embed $g_1^{f(x_1, x_2)} \bmod N = g_1^{a_1 x_1^{real} + a_2 x_2^{real} + b} \bmod N$ into an authentication tag t . Note that if $a_2/a_1 = w \bmod \phi(N)/4$, then $a_1 x_1^{real} + a_2 x_2^{real}$ is linearly dependent with $(x_1^{real} + wx_2^{real})$, and $f(x_1, x_2)$ is not randomly distributed. Fortunately, we prove that this case implies the breaking of the discrete logarithm assumption. If $a_2/a_1 \neq w \bmod \phi(N)/4$, then $a_1 x_1^{real} + a_2 x_2^{real}$ is linearly independent with $(x_1^{real} + wx_2^{real})$, and $a_1 x_1^{real} + a_2 x_2^{real}$ is randomly distributed. In order to place a valid key-dependent decryption query, the adversary would have to guess the value of $a_1 x_1^{real} + a_2 x_2^{real}$.

The second part of the entropy is used to protect an original key k^* , which is used to derive keys for the authenticated encryption by using affine functions $f(k^*) = rk^* + s$. We show that the decryption oracle will not leak any information of (x_1^{hide}, x_2^{hide}) . Thus k^* is perfectly hidden from the adversary. According to the RKA security of the authenticated encryption, the plaintext messages are perfectly protected by the authenticated encryption.

Theorem 2. *Assume the DCR assumption holds in Z_{N^s} , $\bar{\pi}$ is an RKA secure authenticated encryption scheme with respect to affine functions $f(k) = ak + b$ that $a \neq 0$. Then the scheme above is n -KDM-CCA secure with respect to affine functions with the range of $[N^{s-1}]$ for $s \geq 2$.*

The proof is via a sequence of games involving the challenger \mathcal{C} and the adversary \mathcal{A} . Let W_δ be the event that \mathcal{A} guesses b correctly in Game δ .

- **Game 0:** This game is the actual n -KDM-CCA game. The challenger \mathcal{C} runs the setup and key generation algorithm, sends the parameters $prm = (g_1, g_2, N)$ and public keys $pk_i = (h_i = g_1^{x_{i1}} g_2^{x_{i2}}), i = 1, \dots, n$ to the adversary \mathcal{A} . When responding to a key-dependent encryption query with (f_λ, i) , that $f_\lambda(sk_1, \dots, sk_n) = \sum_{j=1}^n (a_{j\lambda 1}x_{j1} + a_{j\lambda 2}x_{j2}) + b_\lambda \in [N^{s-1}]$, the challenger \mathcal{C} randomly chooses $b \in \{0, 1\}$ and computes as follows:

$$\begin{aligned} r_\lambda^*, \tilde{r}_\lambda^* &\xleftarrow{R} [\lfloor N/4 \rfloor], k_\lambda^* \xleftarrow{R} Z_N, \\ m_{\lambda 0} &\leftarrow 0, m_{\lambda 1} \leftarrow \sum_{j=1}^n (a_{j\lambda 1}x_{j1} + a_{j\lambda 2}x_{j2}) + b_\lambda, \\ u_{\lambda 1}^* &\leftarrow g_1^{r_\lambda^*} \mod N^2, u_{\lambda 2}^* \leftarrow g_2^{r_\lambda^*} \mod N^2, e_\lambda^* \leftarrow h_i^{r_\lambda^*} T^{k_\lambda^*} \mod N^2, \\ \tilde{u}_{\lambda 1}^* &\leftarrow g_1^{\tilde{r}_\lambda^*} \mod N^s, \tilde{u}_{\lambda 2}^* \leftarrow g_2^{\tilde{r}_\lambda^*} \mod N^s, \tilde{e}_\lambda^* \leftarrow h_i^{\tilde{r}_\lambda^*} T^{m_{\lambda b}} \mod N^s, \\ t_\lambda^* &\leftarrow H(u_{\lambda 1}^* || u_{\lambda 2}^* || e_\lambda^* || (g_1^{m_{\lambda b}} \mod N)), v_\lambda^* \leftarrow \bar{\pi}.E_{k_\lambda^*}(t_\lambda^* || \tilde{u}_{\lambda 1}^* || \tilde{u}_{\lambda 2}^* || \tilde{e}_\lambda^*), \\ c_\lambda^* &\leftarrow (u_{\lambda 1}^*, u_{\lambda 2}^*, e_\lambda^*, v_\lambda^*). \end{aligned}$$

When responding to a decryption query $(c = (u_1, u_2, e, v), i)$ the challenger \mathcal{C} decrypts using the secret key $sk_i = (x_{i1}, x_{i2})$. By definition we have:

$$\text{AdvKDM}_{\mathcal{A}, n}^{\text{cca}} = \Pr[W_0] - 1/2. \quad (13)$$

- **Game 1:** This game is exactly like Game 0, except that the key generation algorithm runs as follows:

$$\begin{aligned} x_1, x_2 &\xleftarrow{R} [\lfloor N^2/4 \rfloor], \bar{x}_{i1}, \bar{x}_{i2} \xleftarrow{R} [\lfloor N/4 \rfloor], \\ x_{i1} &\leftarrow x_1 + \bar{x}_{i1}, x_{i2} \leftarrow x_1 + \bar{x}_{i2}, h_i \leftarrow g_1^{-x_{i1}} g_2^{-x_{i2}}, \\ pk_i &\leftarrow (h_i), sk_i \leftarrow (x_{i1}, x_{i2}), \end{aligned}$$

It is clear that in the key generation algorithm above x_{i1}, x_{i2} are chosen from $[\lfloor N^2/4 \rfloor + \lfloor N/4 \rfloor]$ instead of $[\lfloor N^2/4 \rfloor]$. The statistical distance between the

uniform distribution over these two domains is about 2^{-l_N} , where l_N is the length of N .

If we use $\phi(N)/4$ instead of $\lfloor N/4 \rfloor$ then, the distributions of the public keys pk_1, \dots, pk_n are identical in Game 0 and Game 1. Since $\phi(N)/4 = pq, \lfloor N/4 \rfloor = pq + (p+q)/2$, hence the statistical distance between the public keys in Game 0 and Game 1 is bounded by $2^{-l_N/2}$.

As a result, we have:

$$\Pr[W_0] \leq \Pr[W_1] + 2^{-l_N} + 2^{-l_N/2}. \quad (14)$$

- **Game 2:** This game is exactly like Game 1, except that when responding to the key-dependent encryption query, \tilde{e}_λ^* and e_λ^* are computed as:

$$e_\lambda^* \leftarrow u_{\lambda 1}^{*-x_{i1}} u_{\lambda 2}^{*-x_{i2}} T^{k_\lambda^*} \mod N^2, \tilde{e}_\lambda^* \leftarrow \tilde{u}_{\lambda 1}^{*-x_{i1}} \tilde{u}_{\lambda 2}^{*-x_{i2}} T^{m_{\lambda b}} \mod N^s.$$

It is clear that $\tilde{u}_{\lambda 1}^{*-x_{i1}} \tilde{u}_{\lambda 2}^{*-x_{i2}} = h_i^{\tilde{r}_\lambda^*}, u_{\lambda 1}^{*-x_{i1}} u_{\lambda 2}^{*-x_{i2}} = h_i^{r_\lambda^*}$. Hence we have:

$$\Pr[W_1] = \Pr[W_2]. \quad (15)$$

- **Game 3:** Game 3 is exactly like Game 2, except that when responding to the key-dependent encryption query, if $b = 1$, then $\tilde{u}_{\lambda 1}^*, \tilde{u}_{\lambda 2}^*, \tilde{e}_\lambda^*$ are computed as follows:

$$\begin{aligned} \tilde{u}_{\lambda 1}^* &\leftarrow g_1^{\tilde{r}_\lambda^*} T^{a_{\lambda 1}} \mod N^s, \\ \tilde{u}_{\lambda 2}^* &\leftarrow g_2^{\tilde{r}_\lambda^*} T^{a_{\lambda 2}} \mod N^s, \\ \tilde{e}_\lambda^* &\leftarrow h_i^{\tilde{r}_\lambda^*} T^{b_\lambda + \rho_\lambda} \mod N^s, \end{aligned}$$

where $a_{\lambda 1} = \sum_{j=1}^n a_{j\lambda 1}$, $a_{\lambda 2} = \sum_{j=1}^n a_{j\lambda 2}$ and $\rho_\lambda = \sum_{j=1}^n (a_{j\lambda 1}(\bar{x}_{j1} - \bar{x}_{i1}) + a_{j\lambda 2}(\bar{x}_{j2} - \bar{x}_{i2}))$.

It is clear that if the adversary \mathcal{A} can distinguish Game 3 from Game 2, then \mathcal{C} can solve the IV2 problem. Concretely, when receiving g_1, g_2 from the IV2 challenger, \mathcal{C} runs the key generation algorithm to get pk_i, sk_i . When responding to the key-dependent encryption query, if $b = 1$, $\tilde{u}_{\lambda 1}^*, \tilde{u}_{\lambda 2}^*, \tilde{e}_\lambda^*$ are computed as follows:

$$(\tilde{u}_{\lambda 1}^*, \tilde{u}_{\lambda 2}^*) \leftarrow \mathcal{O}_{iv2}(a_{\lambda 1}, a_{\lambda 2}), \tilde{e}_\lambda^* \leftarrow \tilde{u}_{\lambda 1}^{*-x_{i1}} \tilde{u}_{\lambda 2}^{*-x_{i2}} T^{m_{\lambda b}} \mod N^s.$$

Other parts of the ciphertext and the response to the decryption queries are computed as in Game 2. According to the definition of the IV2 problem, $(\tilde{u}_{\lambda 1}^*, \tilde{u}_{\lambda 2}^*) = (g_1^{\tilde{r}_\lambda^*}, g_2^{\tilde{r}_\lambda^*})$ or $(\tilde{u}_{\lambda 1}^*, \tilde{u}_{\lambda 2}^*) = (g_1^{\tilde{r}_\lambda^*} T^{a_{\lambda 1}}, g_2^{\tilde{r}_\lambda^*} T^{a_{\lambda 2}})$. It is easy to verify that in the first case the response to the key-dependent encryption query is identical to that of Game 2. In the second case we have:

$$\begin{aligned} \tilde{e}_\lambda^* &= \tilde{u}_{\lambda 1}^{*-x_{i1}} \tilde{u}_{\lambda 2}^{*-x_{i2}} T^{m_{\lambda 1}} \\ &= (g_1^{\tilde{r}_\lambda^*} T^{a_{\lambda 1}})^{-x_{i1}} (g_2^{\tilde{r}_\lambda^*} T^{a_{\lambda 2}})^{-x_{i2}} T^{\sum_{j=1}^n (a_{j\lambda 1}x_{j1} + a_{j\lambda 2}x_{j2}) + b_\lambda} \\ &= (g_1^{\tilde{r}_\lambda^*} T^{\sum_{j=1}^n a_{j\lambda 1}})^{-x_{i1}} (g_2^{\tilde{r}_\lambda^*} T^{\sum_{j=1}^n a_{j\lambda 2}})^{-x_{i2}} T^{\sum_{j=1}^n (a_{j\lambda 1}x_{j1} + a_{j\lambda 2}x_{j2}) + b_\lambda} \\ &= h^{\tilde{r}_\lambda^*} T^{\sum_{j=1}^n (a_{j\lambda 1}(x_{j1} - x_{i1}) + a_{j\lambda 2}(x_{j2} - x_{i2})) + b_\lambda} \\ &= h^{\tilde{r}_\lambda^*} T^{\sum_{j=1}^n (a_{j\lambda 1}(\bar{x}_{j1} - \bar{x}_{i1}) + a_{j\lambda 2}(\bar{x}_{j2} - \bar{x}_{i2})) + b_\lambda} \\ &= h^{\tilde{r}_\lambda^*} T^{\rho_\lambda + b_\lambda} \end{aligned}$$

Hence we have:

$$\Pr[W_2] \leq \Pr[W_3] + \text{Adv}_{\mathcal{A}}^{\text{iv2}}. \quad (16)$$

- **Game 4:** This game is exactly like Game 3, except that when responding to the key-dependent encryption query, the challenger \mathcal{C} randomly chooses $\alpha, \beta \in Z_N, r^* \in [\lfloor N/4 \rfloor]$, computes $u_{\lambda 1}^*, u_{\lambda 2}^*, e_{\lambda}^*$ as follows:

$$\begin{aligned} u_{\lambda 1}^* &\leftarrow (g_1^{r^*} T^\alpha)^{r_{\lambda}^*} \mod N^2, \\ u_{\lambda 2}^* &\leftarrow (g_2^{r^*} T^\beta)^{r_{\lambda}^*} \mod N^2, \\ e_{\lambda}^* &\leftarrow (u_{\lambda 1}^{*-x_{i1}} u_{\lambda 2}^{*-x_{i2}}) T^{k_{\lambda}^*} \mod N^2, \end{aligned}$$

Similar as in Game 3, if the adversary \mathcal{A} can distinguish Game 4 from Game 3, then \mathcal{C} can solve the IV2 problem. Hence we have:

$$\Pr[W_3] \leq \Pr[W_4] + \text{Adv}_{\mathcal{A}}^{\text{iv2}}. \quad (17)$$

- **Game 5:** This game is exactly like Game 4, except that when responding to the key-dependent encryption query, the challenger \mathcal{C} randomly chooses $k^* \in Z_N$, computes k_{λ}^* as follows:

$$s_{\lambda}^* \xleftarrow{R} Z_N, k_{\lambda}^* \leftarrow r_{\lambda}^* k^* + s_{\lambda}^* \mod N.$$

It is to verify that $k_{\lambda}^* = r_{\lambda}^* k^* + s_{\lambda}^* \mod N$ is uniformly distributed on Z_N . Hence we have:

$$\Pr[W_4] = \Pr[W_5]. \quad (18)$$

- **Game 6:** This game is exactly like Game 5, except that when responding to the decryption query $(c = (u_1, u_2, e, v), i)$, the challenger \mathcal{C} computes k and m by using $\phi(N) = (p-1)(q-1)$ and $sk_i = (x_{i1}, x_{i2})$ as follows:

$$\begin{aligned} \alpha' &\leftarrow \text{dlog}_T(u_1^{\phi(N)})/\phi(N) \mod N, \beta' \leftarrow \text{dlog}_T(u_2^{\phi(N)})/\phi(N) \mod N, \\ \gamma' &\leftarrow \text{dlog}_T(e^{\phi(N)})/\phi(N) \mod N, k \leftarrow (\alpha' x_{i1} + \beta' x_{i2} + \gamma') \mod N, \\ \tilde{\alpha} &\leftarrow \text{dlog}_T(\tilde{u}_1^{\phi(N)})/\phi(N) \mod N^{s-1}, \tilde{\beta} \leftarrow \text{dlog}_T(\tilde{u}_2^{\phi(N)})/\phi(N) \mod N^{s-1}, \\ \tilde{\gamma} &\leftarrow \text{dlog}_T(\tilde{e}^{\phi(N)})/\phi(N) \mod N^{s-1}, m \leftarrow (\tilde{\alpha} x_{i1} + \tilde{\beta} x_{i2} + \tilde{\gamma}) \mod N^{s-1}. \end{aligned}$$

It is clear that, the computation of the decryption algorithm in Game 6 is identical to that in Game 5, we have:

$$\Pr[W_5] = \Pr[W_6]. \quad (19)$$

- **Game 7:** This game is exactly like Game 6, except that when responding to the decryption query, the challenger returns \perp when $u_1 = u_{\lambda 1}^*, u_2 = u_{\lambda 2}^*, e = e_{\lambda}^*, v \neq v_{\lambda}^*$. Since $\bar{\pi}$ is INT-RKA secure and k^* is randomly distributed from the point view of \mathcal{A} (see Game 8 equation 23 for detailed analysis), such ciphertext will be rejected except with the probability of $\text{Adv}_{\mathcal{A}}^{\text{int-rka}}$. Hence we have:

$$\Pr[W_6] \leq \Pr[W_7] + \text{Adv}_{\mathcal{A}}^{\text{int-rka}}. \quad (20)$$

- **Game 8:** This game is exactly like Game 7, except that when responding to the decryption query, the challenger returns \perp when $\tilde{\alpha} \neq 0$ or $\tilde{\beta} \neq 0$ or $\alpha' \neq 0$ or $\beta' \neq 0$.

Let $x_1^{hide} = x_1 \bmod N, x_1^{real} = x_1 \bmod \phi(N)/4, x_2^{hide} = x_2 \bmod N, x_2^{real} = x_2 \bmod \phi(N)/4, x_{i1}^{hide} = x_{i1} \bmod N, x_{i1}^{real} = x_{i1} \bmod \phi(N)/4, x_{i2}^{hide} = x_{i2} \bmod N, x_{i2}^{real} = x_{i2} \bmod \phi(N)/4, w = \log_{g_1} g_2$, according to the key generation algorithm we have:

$$\begin{aligned} \log_{g_1} h_i &= -(x_{i1}^{real} + wx_{i2}^{real}) \bmod \phi(N)/4 \\ &= -(x_1^{real} + \bar{x}_{i1}^{real} + w(x_2^{real} + \bar{x}_{i2}^{real})) \bmod \phi(N)/4. \end{aligned} \quad (21)$$

Hence, x_1^{hide} and x_2^{hide} are randomly distributed from the point view of \mathcal{A} conditioned on the public key. According to the encryption oracle k^* is encapsulated in e_λ^* as follows:

$$\begin{aligned} e_\lambda^* &= (u_{\lambda 1}^{*-x_{i1}} u_{\lambda 2}^{*-x_{i2}}) T^{k_\lambda^*} \bmod N^2 \\ &= ((g_1^{r_\lambda^*} T^\alpha)^{r_\lambda^*})^{-x_{i1}} ((g_2^{r_\lambda^*} T^\beta)^{r_\lambda^*})^{-x_{i2}} T^{(r_\lambda^* k^* + s_\lambda^*)} \bmod N^2 \\ &= h_i^{r_\lambda^* r_\lambda^*} T^{r_\lambda^* (-\alpha x_{i1} - \beta x_{i2} + k^*) + s_\lambda^*} \bmod N^2 \\ &= h_i^{r_\lambda^* r_\lambda^*} T^{r_\lambda^* (-\alpha(x_1 + \bar{x}_{i1}) - \beta(x_2 + \bar{x}_{i2}) + k^*) + s_\lambda^*} \bmod N^2 \end{aligned} \quad (22)$$

$$\begin{aligned} k^* &= \frac{\text{dlog}_T(e_\lambda^{*\phi(N)})/\phi(N)}{r_\lambda^*} + \alpha(x_1 + \bar{x}_{i1}) + \beta(x_2 + \bar{x}_{i2}) - \frac{s_\lambda^*}{r_\lambda^*} \bmod N \\ &= \frac{\text{dlog}_T(e_\lambda^{*\phi(N)})/\phi(N)}{r_\lambda^*} + \alpha(x_1^{hide} + \bar{x}_{i1}^{hide}) + \beta(x_2^{hide} + \bar{x}_{i2}^{hide}) - \frac{s_\lambda^*}{r_\lambda^*} \bmod N \\ &= \frac{\text{dlog}_T(e_\lambda^{*\phi(N)})/\phi(N)}{r_\lambda^*} + \alpha x_1^{hide} + \beta x_2^{hide} + \alpha \bar{x}_{i1}^{hide} + \beta \bar{x}_{i2}^{hide} - \frac{s_\lambda^*}{r_\lambda^*} \bmod N. \end{aligned} \quad (23)$$

It is clear that, when $\tilde{\alpha} = 0, \tilde{\beta} = 0, \alpha' = 0, \beta' = 0$, the decryption oracle will not leak any information of the private key. In addition ciphertexts that $u_1 = u_{\lambda 1}^*, u_2 = u_{\lambda 2}^*, e = e_\lambda^*, v \neq v_\lambda^*$ are rejected. Denote Bad as the event that the challenge does not return \perp when $\tilde{\alpha} \neq 0$ or $\tilde{\beta} \neq 0$ or $\alpha' \neq 0$ or $\beta' \neq 0$. We have that if Bad does not happen, the decryption will not leak any information of k^* . Hence k^* is randomly distributed from the point view of \mathcal{A} conditioned on $\neg Bad$. Now we show that, in Game 7 when $\tilde{\alpha} \neq 0$ or $\tilde{\beta} \neq 0$ or $\alpha' \neq 0$ or $\beta' \neq 0$, the challenger will return \perp except with a negligible probability. For clarity, we consider four cases as follows:

- $\tilde{\alpha} \neq 0, \tilde{\beta} \neq 0$: Since k^* is randomly distributed conditioned on $\neg Bad$ and the keys $k_\lambda^* = r_\lambda^* k^* + s_\lambda^*$ are affine functions of k^* , according to the IND-RKA security of $\bar{\pi}$ we have that v_λ^* will not leak any information of x_1^{real} and x_2^{real} except with negligible probability of $\text{Adv}_{\mathcal{A}}^{\text{ind-rka}}$. Hence the only information that the adversary gets about x_1^{real} and x_2^{real} conditioned on the public key and the ciphertexts is equation (21). If $\tilde{\beta}/\tilde{\alpha} \neq w \bmod \phi(N)/4$, we have that $\log_{g_1} g_1^m = \tilde{\alpha}(x_1^{real} + \bar{x}_{i1}^{real}) + \tilde{\beta}(x_2^{real} + \bar{x}_{i2}^{real}) + \tilde{\gamma} \bmod \phi(N)/4$ is linearly independent with equation (21). Thus $t = H(u_1||u_2||e||g_1^m)$ is randomly distributed from the point view of the adversary \mathcal{A} except with negligible probability of $\text{Adv}_{\mathcal{A}}^{\text{ind-rka}}$.

So we have that such ciphertexts will be rejected except with the probability of $\text{Adv}_{\mathcal{A}}^{\text{ind-rka}} + 2^{-l_t}$.

If $\tilde{\beta}/\tilde{\alpha} = w \pmod{\phi(N)}$ the challenger \mathcal{C} can compute:

$$w = \tilde{\beta}/\tilde{\alpha} = \text{dlog}_T(\tilde{u}_2^{\phi(N)})/\text{dlog}_T(\tilde{u}_1^{\phi(N)}).$$

Let ϵ_{dlg} be the probability that any adversary breaks the discrete logarithm assumption, we have that the probability that such cases happens is ϵ_{dlg} .

- $\tilde{\alpha} \neq 0, \tilde{\beta} = 0$ or $\tilde{\alpha} = 0, \tilde{\beta} \neq 0$: Similar as the case $\tilde{\beta}/\tilde{\alpha} \neq w \pmod{\phi(N)/4}$ above, such ciphertexts will be rejected except with the probability of $\text{Adv}_{\mathcal{A}}^{\text{ind-rka}} + 2^{-l_t}$.
- $\alpha' \neq 0, \beta' \neq 0$: If $\alpha'/\beta' \neq \alpha/\beta$, we have that $k = \alpha'(x_1^{\text{hide}} + \bar{x}_{i1}^{\text{hide}}) + \beta'(x_2^{\text{hide}} + \bar{x}_{i2}^{\text{hide}}) + \gamma' \pmod{N}$ is linearly independent with equation (23). Hence $k \in Z_N$ is randomly distributed from the point view of \mathcal{A} , such ciphertexts will be rejected except with the probability of 2^{-l_N} . If $\alpha'/\beta' = \alpha/\beta \pmod{N}$, let $\alpha' = \bar{r}\alpha \pmod{N}, \beta' = \bar{r}\beta \pmod{N}, \gamma' = \bar{r}(-\alpha x_{i1}^{\text{hide}} - \beta x_{i2}^{\text{hide}} + k^*) + \gamma \pmod{N}$, we have:

$$k = \bar{r}k^* + \gamma.$$

According to the INT-RKA security of $\bar{\pi}$, such ciphertexts will be rejected except with the probability of $\text{Adv}_{\mathcal{A}}^{\text{int-rka}}$.

- $\alpha' \neq 0, \beta' = 0$ or $\alpha' = 0, \beta' \neq 0$: Similar as the case of $\alpha'/\beta' \neq \alpha/\beta \pmod{N}$, such ciphertexts will be rejected except with the probability of 2^{-l_N} .

According to the analysis above, we have that in Game 7:

$$\Pr[Bad] \leq \text{Adv}_{\mathcal{A}}^{\text{ind-rka}} + \text{Adv}_{\mathcal{A}}^{\text{int-rka}} + 2^{-l_t} + 2^{-l_N} + \epsilon_{\text{dlg}}$$

Hence we have:

$$\Pr[W_7] \leq \Pr[W_8] + \Pr[Bad]. \quad (24)$$

- **Game 9:** This game is exactly like Game 8, except that when responding to the key-dependent encryption query, the challenger \mathcal{C} randomly chooses $k^*, \bar{k}^* \in Z_N$, computes e_{λ}^* and v_{λ}^* as follows:

$$\begin{aligned} s_{\lambda}^* &\xleftarrow{R} Z_N, k_{\lambda}^* \leftarrow r_{\lambda}^* k^* + s_{\lambda}^* \pmod{N}, e_{\lambda}^* \leftarrow (u_{\lambda 1}^{*-x_{i1}} u_{\lambda 2}^{*-x_{i2}}) T^{k_{\lambda}^*} \pmod{N^2}, \\ \bar{k}_{\lambda}^* &\leftarrow r_{\lambda}^* \bar{k}^* + s_{\lambda}^* \pmod{N}, v_{\lambda}^* \leftarrow \bar{\pi}.E_{\bar{k}_{\lambda}^*}(t_{\lambda}^* || \tilde{u}_{\lambda 1}^* || \tilde{u}_{\lambda 2}^* || \tilde{e}_{\lambda}^*). \end{aligned}$$

Since all the decryption queries that $\tilde{\alpha} \neq 0$ or $\tilde{\beta} \neq 0$ or $\alpha' \neq 0$ or $\beta' \neq 0$ are rejected, we have that x_1^{hide} and x_2^{hide} are randomly distributed from the point view of \mathcal{A} conditioned on the public key and the decryption oracle. In addition, when $(u_1 = u_{\lambda 1}^*, u_2 = u_{\lambda 2}^*, e = e_{\lambda}^*, v \neq v_{\lambda}^*)$ decryption queries are also rejected. Hence, the decryption oracle will not leak any information of k^* . According to equation (23), k^* is randomly distributed. So we have:

$$\Pr[W_8] \leq \Pr[W_9]. \quad (25)$$

According to the IND-RKA security of $\bar{\pi}$, the probability that \mathcal{A} wins in Game 9 is:

$$\Pr[W_9] \leq 1/2 + \text{Adv}_{\mathcal{A}}^{\text{ind-rka}}. \quad (26)$$

According to equations above we have:

$$\text{AdvKDM}_{\mathcal{A},n}^{\text{cca}} \leq 2\text{Adv}_{\mathcal{A}}^{\text{iv2}} + 2\text{Adv}_{\mathcal{A}}^{\text{int-rka}} + 2\text{Adv}_{\mathcal{A}}^{\text{ind-rka}} + \epsilon. \quad (27)$$

where $\epsilon = 2 \cdot 2^{-l_N} + 2^{-l_N/2} + 2^{-l_t} + \epsilon_{\text{dlg}}$.

This completes the proof of the theorem 2. \square

5 Conclusion

We propose an efficient KDM-CCA secure public key encryption scheme with respect to affine functions by enhancing an IND-CCA2 secure hybrid encryption scheme based on the high entropy hash proof system. Our main idea is to divide the entropy of the private key into two parts: one part is embedded into the authentication tag, the other part is used to protect an original key for the authenticated encryption scheme. To hide the authentication tags from the adversary perfectly, we use an RKA secure authenticated encryption scheme and derive the keys from affine functions of the original key.

Compared with Hofheinz's scheme [33], our new scheme is simpler and more efficient. In addition, our new scheme achieves KDM-CCA security with respect to affine functions while Hofheinz's scheme only achieves CIRC-CCA security.

References

1. Abdalla, M., Benhamouda, F., Passelègue, A., Paterson, K.G.: Related-key security for pseudorandom functions beyond the linear barrier. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 77–94. Springer, Heidelberg (2014)
2. Alperin-Sheriff, J., Peikert, C.: Circular and KDM security for identity-based encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 334–352. Springer, Heidelberg (2012)
3. Applebaum, B.: Key-dependent message security: Generic amplification and completeness. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 527–546. Springer, Heidelberg (2011)
4. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009)
5. Applebaum, B., Harnik, D., Ishai, Y.: Semantic security under related-key attacks and applications. In: Proceedings of Innovations in Computer Science - ICS 2010, pp. 45–60. Tsinghua University, Beijing, January 7–9 (2011)
6. Backes, M., Dürmuth, M., Unruh, D.: OAEP is secure under key-dependent messages. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 506–523. Springer, Heidelberg (2008)

7. Backes, M., Pfitzmann, B., Scedrov, A.: Key-dependent message security under active attacks - brsim/uc-soundness of dolev-yao-style encryption with key cycles. *Journal of Computer Security* **16**(5), 497–530 (2008)
8. Barak, Boaz, Haitner, Iftach, Hofheinz, Dennis, Ishai, Yuval: Bounded Key-Dependent Message Security. In: Gilbert, Henri (ed.) *EUROCRYPT 2010*. LNCS, vol. 6110, pp. 423–444. Springer, Heidelberg (2010)
9. Bellare, Mihir, Cash, David: Pseudorandom Functions and Permutations Provably Secure against Related-Key Attacks. In: Rabin, Tal (ed.) *CRYPTO 2010*. LNCS, vol. 6223, pp. 666–684. Springer, Heidelberg (2010)
10. Bellare, Mihir, Cash, David, Miller, Rachel: Cryptography Secure against Related-Key Attacks and Tampering. In: Lee, Dong Hoon, Wang, Xiaoyun (eds.) *ASIACRYPT 2011*. LNCS, vol. 7073, pp. 486–503. Springer, Heidelberg (2011)
11. Bellare, M., Keelveedhi, S.: Authenticated and Misuse-Resistant Encryption of Key-Dependent Data. In: Rogaway, P. (ed.) *CRYPTO 2011*. LNCS, vol. 6841, pp. 610–629. Springer, Heidelberg (2011)
12. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: Rka-prps, In: Biham, E. (ed.) *EUROCRYPT 2003*. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (2003)
13. Bellare, M., Paterson, K.G., Thomson, S.: RKA security beyond the linear barrier: IBE, encryption and signatures. In: Sako, K., Wang, X. (eds.) *ASIACRYPT 2012*. LNCS, vol. 7658, pp. 331–348. Springer, Heidelberg (2012)
14. Biham, E.: New types of cryptanalytic attacks using related keys. In: Helleseth, T. (ed.) *EUROCRYPT 1993*. LNCS, vol. 765, pp. 398–409. Springer, Heidelberg (1994)
15. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) *CRYPTO 1997*. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
16. Black, J., Rogaway, P., Shrimpton, T.: Encryption-scheme security in the presence. In: Nyberg, K., Heys, H.M. (eds.) *SAC 2002*. LNCS, vol. 2595, pp. 62–75. Springer, Heidelberg (2003)
17. Böhl, F., Davies, G.T., Hofheinz, D.: Encryption schemes secure under related-key and key-dependent message attacks. In: Krawczyk, H. (ed.) *PKC 2014*. LNCS, vol. 8383, pp. 483–500. Springer, Heidelberg (2014)
18. 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)
19. Boneh, D., Halevi, S., Hamburg, M., Ostrovsky, R.: Circular-secure encryption from decision diffie-hellman. In: Wagner, D. (ed.) *CRYPTO 2008*. LNCS, vol. 5157, pp. 108–125. Springer, Heidelberg (2008)
20. Brakerski, Z., Goldwasser, S.: Circular and leakage resilient public-key encryption under subgroup indistinguishability. In: Rabin, T. (ed.) *CRYPTO 2010*. LNCS, vol. 6223, pp. 1–20. Springer, Heidelberg (2010)
21. Brakerski, Z., Goldwasser, S., Kalai, Y.T.: Black-box circular-secure encryption beyond affine functions. In: Ishai, Y. (ed.) *TCC 2011*. LNCS, vol. 6597, pp. 201–218. Springer, Heidelberg (2011)
22. Camenisch, J., Chandran, N., Shoup, V.: A public key encryption scheme secure against key dependent chosen plaintext and adaptive chosen ciphertext attacks. In: Joux, A. (ed.) *EUROCRYPT 2009*. LNCS, vol. 5479, pp. 351–368. Springer, Heidelberg (2009)

23. Camenisch, J.L., Lysyanskaya, A.: An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In: Pfitzmann, B. (ed.) *EUROCRYPT* 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (2001)
24. Canetti, R., Halevi, S., Katz, J.: Chosen-Ciphertext Security from Identity-Based Encryption. In: Cachin, C., Camenisch, J.L. (eds.) *EUROCRYPT* 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
25. Cramer, R., Shoup, V.: Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In: Knudsen, L.R. (ed.) *EUROCRYPT* 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (2002)
26. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier’s probabilistic public-key system. In: Kim, K., (ed.) *PKC* 2001. LNCS 1992, pp. 119–136. Springer, Heidelberg (2001)
27. Galindo, D., Herranz, J., Villar, J.: Identity-Based Encryption with Master Key-Dependent Message Security and Leakage-Resilience. In: Foresti, S., Yung, M., Martinelli, F. (eds.) *ESORICS* 2012. LNCS, vol. 7459, pp. 627–642. Springer, Heidelberg (2012)
28. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, pp. 169–178 (2009)
29. Goldenberg, D., Liskov, M.: On Related-Secret Pseudorandomness. In: Micciancio, D. (ed.) *TCC* 2010. LNCS, vol. 5978, pp. 255–272. Springer, Heidelberg (2010)
30. Goyal, V., O’Neill, A., Rao, V.: Correlated-Input Secure Hash Functions. In: Ishai, Y. (ed.) *TCC* 2011. LNCS, vol. 6597, pp. 182–200. Springer, Heidelberg (2011)
31. Haitner, I., Holenstein, T.: On the (Im)Possibility of Key Dependent Encryption. In: Reingold, O. (ed.) *TCC* 2009. LNCS, vol. 5444, pp. 202–219. Springer, Heidelberg (2009)
32. Halevi, S., Krawczyk, H.: Security under key-dependent inputs. In: Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28–31, 2007. pp. 466–475 (2007)
33. Hofheinz, D.: Circular Chosen-Ciphertext Security with Compact Ciphertexts. In: Johansson, T., Nguyen, P.Q. (eds.) *EUROCRYPT* 2013. LNCS, vol. 7881, pp. 520–536. Springer, Heidelberg (2013)
34. Hofheinz, D., Unruh, D.: Towards Key-Dependent Message Security in the Standard Model. In: Smart, N.P. (ed.) *EUROCRYPT* 2008. LNCS, vol. 4965, pp. 108–126. Springer, Heidelberg (2008)
35. Jia, D., Li, B., Lu, X., Mei, Q.: Related key secure PKE from hash proof systems. In: Proceedings of the Advances in Information and Computer Security - 9th International Workshop on Security, IWSEC 2014, Hirosaki, Japan, August 27–29, pp. 250–265 (2014). <http://dx.doi.org/10.1007/978-3-319-09843-2>
36. Jia, D., Lu, X., Li, B., Mei, Q.: RKA Secure PKE Based on the DDH and HR Assumptions. In: Susilo, W., Reyhanitabar, R. (eds.) *ProvSec* 2013. LNCS, vol. 8209, pp. 271–287. Springer, Heidelberg (2013)
37. Kiltz, E., Pietrzak, K., Stam, M., Yung, M.: A New Randomness Extraction Paradigm for Hybrid Encryption. In: Joux, A. (ed.) *EUROCRYPT* 2009. LNCS, vol. 5479, pp. 590–609. Springer, Heidelberg (2009)
38. Knudsen, L.R.: Cryptanalysis of LOKI91. In: Proceedings of the Advances in Cryptology - AUSCRYPT 1992, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13–16, 1992, pp. 196–208 (1992)

39. Lu, X., Li, B., Jia, D.: Related-Key Security for Hybrid Encryption. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) ISC 2014. LNCS, vol. 8783, pp. 19–32. Springer, Heidelberg (2014)
40. Malkin, T., Teranishi, I., Yung, M.: Efficient Circuit-Size Independent Public Key Encryption with KDM Security. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 507–526. Springer, Heidelberg (2011)
41. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13–17, Baltimore, Maryland, USA. pp. 427–437 (1990)
42. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
43. Wee, H.: Public Key Encryption against Related Key Attacks. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 262–279. Springer, Heidelberg (2012)

On the Provable Security of the Iterated Even-Mansour Cipher Against Related-Key and Chosen-Key Attacks

Benoît Cogliati^{1(✉)} and Yannick Seurin²

¹ University of Versailles, Versailles, France

benoitcogliati@hotmail.fr

² ANSSI, Paris, France

yannick.seurin@m4x.org

Abstract. The iterated Even-Mansour cipher is a construction of a block cipher from r public permutations P_1, \dots, P_r which abstracts in a generic way the structure of key-alternating ciphers. The indistinguishability of this construction from a truly random permutation by an adversary with oracle access to the inner permutations P_1, \dots, P_r has been investigated in a series of recent papers. This construction has also been shown to be (fully) indifferentiable from an ideal cipher for a sufficient number of rounds (five or twelve depending on the assumptions on the key-schedule). In this paper, we extend this line of work by considering the resistance of the iterated Even-Mansour cipher to xor-induced related-key attacks (i.e., related-key attacks where the adversary is allowed to xor any constant of its choice to the secret key) and to chosen-key attacks. For xor-induced related-key attacks, we first provide a distinguishing attack for two rounds, assuming the key-schedule is linear. We then prove that for a linear key-schedule, three rounds yield a cipher which is secure against xor-induced related-key attacks up to $\mathcal{O}(2^{\frac{n}{2}})$ queries of the adversary, whereas for a nonlinear key-schedule, one round is sufficient to obtain a similar security bound. We also show that the iterated Even-Mansour cipher with four rounds offers some form of provable resistance to chosen-key attacks, which is the minimal number of rounds to achieve this property. The main technical tool that we use to prove this result is *sequential indifferentiability*, a weakened variant of (full) indifferentiability introduced by Mandal *et al.* (TCC 2010).

Keywords: Block cipher · Ideal cipher · Related-key attacks · Chosen-key attacks · Iterated Even-Mansour cipher · Key-alternating cipher · Indifferentiability · Correlation intractability

1 Introduction

BACKGROUND. The Even-Mansour construction, and its generalization, the iterated Even-Mansour (*IEM* for short) construction, is a very simple way to define

Y. Seurin—This author was partially supported by the French National Agency of Research through the BLOC project (contract ANR-11-INS-011).

a block cipher from a set of r public permutations P_1, \dots, P_r of $\{0, 1\}^n$. Given a plaintext $x \in \{0, 1\}^n$, the ciphertext y is computed as

$$y = k_r \oplus P_r(k_{r-1} \oplus P_{r-1}(\cdots P_2(k_1 \oplus P_1(k_0 \oplus x)) \cdots)),$$

where the n -bit round keys k_0, \dots, k_r are either independent or derived from a master key k through key derivation functions $(\gamma_0, \dots, \gamma_r)$. It abstracts in a generic way the high-level structure of most key-alternating ciphers such as AES. The nonexistence of *generic* attacks (i.e., attacks that are possible independently of a particular instantiation of the permutations P_1, \dots, P_r) against this construction can be studied in the Random Permutation Model, where the P_i 's are modeled as public random permutations to which the adversary is only given black-box (oracle) access.

The security of this construction in the traditional (single-key) indistinguishability framework (in other words, its pseudorandomness) has been extensively studied, starting with the seminal work of Even and Mansour for $r = 1$ round [16]. For an arbitrary number r of rounds, the case where all round keys are independent is by now well understood [8, 11, 23, 32], and a tight security bound of $\mathcal{O}(2^{\frac{rn}{r+1}})$ queries has been established [11]. Chen *et al.* [10] also considered, for $r = 2$, the more complex case where the round keys are derived from an n -bit master key (as well as the case where the two inner permutations P_1 and P_2 are identical), and showed that a $\mathcal{O}(2^{\frac{2n}{3}})$ -security bound still holds in that case.

On the other hand, two recent papers [1, 24] explored a very strong security property of this construction, namely (*full*) *indifferentiability from an ideal cipher* (where “full” indifferentiability refers to the notion of Maurer *et al.* [27]), which roughly ensures that the construction “behaves” in some well-defined sense as an ideal cipher, i.e., a block cipher drawn at random from the set of all block ciphers of some given block- and key-length. Andreeva *et al.* [1] showed that this property is achieved by the 5-round IEM cipher, assuming the key derivation function is modeled as a random oracle, while Lampe and Seurin [24] showed this for the 12-round IEM cipher, lifting the cryptographic requirement on the key derivation (namely, their result holds for the *trivial* key-schedule, i.e., when all round keys are equal to the n -bit master key).

In this paper, we complete the picture of the security of the IEM construction by considering security notions that lie between mere pseudorandomness and full indifferentiability from an ideal cipher, namely security against xor-induced related-key attacks (*XRKA* for short), i.e., related-key attacks where the adversary is allowed to xor any constant of its choice to the secret key, and against chosen-key attacks (*CKA* for short).

RELATED-KEY ATTACKS. We start by considering XRKAs, which are important for at least two reasons. First, they arise naturally in a number of contexts, such as the f8 and f9 protocols of the 3GPP standard [20]. Second, from a theoretical point of view, they are the simplest kind of attacks to have the *completeness* property [18], namely, for any keys $k, k' \in \{0, 1\}^n$, there exists $\Delta \in \{0, 1\}^n$ such

that $k \oplus \Delta = k'$. In order to study the resistance of the r -round IEM cipher to XRKAs, we use the traditional indistinguishability-based model of Bellare and Kohno [4], albeit adapted to the Random Permutation Model. This means that the adversary has access to $r + 1$ oracles: a *related key oracle* which takes as input an offset $\Delta \in \{0,1\}^n$ and a plaintext $x \in \{0,1\}^n$ (or a ciphertext $y \in \{0,1\}^n$), and r permutation oracles that we denote $P = (P_1, \dots, P_r)$. The goal of the adversary is to distinguish two worlds: the “real” world, where on input (Δ, x) , the related key oracle returns $\text{EM}_{k \oplus \Delta}^P(x)$, where EM^P is the iterated Even-Mansour construction instantiated with permutations P and $k \in \{0,1\}^n$ is a random key, and the “ideal” world, where the related key oracle returns $E_{k \oplus \Delta}(x)$ for a random block cipher E independent from P . We start by describing a very efficient distinguishing XPKA on the 2-round IEM construction whenever the key derivation functions γ_i are linear (with respect to xor).¹ This somehow comes as a surprise since Bogdanov *et al.* [8] had previously conjectured that two rounds should be sufficient to prevent “certain types” of related-key attacks.² Motivated by this finding, we then consider what is the minimal number of rounds required to achieve provable security against XRKAs.³ We first show that for the trivial key-schedule (all round keys are equal to the n -bit master key), the 3-round IEM cipher is secure against XRKAs up to $\mathcal{O}(2^{\frac{n}{2}})$ queries of the adversary. We conjecture that this bound is tight, but we were unable to find a matching attack (we also conjecture that a matching attack must be adaptive and make two-sided queries to the related-key oracle). If one is willing to use a cryptographically strong key-schedule, we show that a similar security bound is already attained with one round, assuming the key derivation functions are nonlinear (i.e., they have a small maximal differential probability). In this latter case, we note that our security bound is matched by a standard (i.e., non related-key) attack, namely Daemen’s attack [13].

CHOSEN-KEY ATTACKS. We then turn our attention to an even stronger adversarial setting, namely chosen-key attacks [6, 22]. In this model, the adversary is given a block cipher, and its goal is, very informally, to exhibit some non-random behavior of the cipher, for keys and plaintext/ciphertext pairs of its choice. Rigorously formalizing what a non-random behavior means without ending with

¹ Usually, in the case of standard (single-key) attacks, a distinguishing attack immediately gives rise to a key-recovery attack with similar complexity. This does not seem to be the case here, and we do not know whether our distinguishing XPKA can be converted into a key-recovery XPKA of similar complexity.

² The authors of [8] did not formulate any precise conjecture, but they mention that the best related-key attack they are aware of for two rounds and identical round keys is a *key-recovery* attack requiring $\mathcal{O}(2^{\frac{n}{2}})$ queries (see Appendix C.3 of the full version of their paper). Our own attack does not really improve on Bogdanov *et al.*’s one since it is a distinguishing attack, yet it implies that two rounds cannot be deemed secure against XRKAs.

³ We only consider the case where all round keys are derived from the same n -bit master key k . Indeed, it is not hard to see that when round keys are independent, there are trivial XRKAs [8].

an unachievable definition turns out to be elusive for similar reasons that it is hard to rigorously define what collision resistance means for a single hash function [9, 31].⁴ Luckily, working in the Random Permutation Model allows us to escape those complications since it is somehow equivalent to considering a *large class of ciphers* consisting of all key-alternating ciphers of a given block-length and with a given key-schedule (rather than a single fully specified one, say, AES-128). In this setting, we are able to rigorously define resistance to CKAs thanks to the notion of *correlation intractability* first introduced by Canetti *et al.* [9] in the context of hash functions.

The most convenient way we are aware of to prove that a block cipher construction is correlation intractable is to use a weakened variant of “full” indifferentiability [27], named *sequential indifferentiability* (seq-indifferentiability for short), introduced by Mandal *et al.* [26] to prove that the 6-round Feistel construction is correlation intractable. In a nutshell, a block cipher construction \mathcal{C}^F based on an underlying ideal primitive F is (fully) indifferentiable from an ideal cipher if there exists a simulator \mathcal{S} such that the two systems (\mathcal{C}^F, F) , where F is random, and (E, \mathcal{S}^E) , where E is an ideal (random) cipher, are indistinguishable by any (polynomially bounded) adversary \mathcal{D} . The distinguisher can query its two oracles as it wishes, and in the ideal world (E, \mathcal{S}^E) , the simulator is not aware of the queries made by \mathcal{D} directly to E . Seq-indifferentiability is defined as full indifferentiability, except that the distinguisher is restricted to only query its right oracle in a first phase (F or \mathcal{S}^E), and then only its left oracle (\mathcal{C}^F or E). Seq-indifferentiability is closely related to the notion of public indifferentiability [14, 34], where in the ideal world the simulator gets to know all the queries of the distinguisher to the ideal primitive (i.e., the ideal cipher E in our context). We first give a “composition” theorem which relates seq-indifferentiability and correlation intractability (a similar one was already proved in [26], but here we explicitly relate the various parameters since it is important for concrete security statements). Then, we prove that the 4-round IEM cipher, with the trivial key-schedule, is seq-indifferentiable from an ideal cipher (by a previous attack by Lampe and Seurin [24], this is also the minimal number of rounds to obtain this property). This implies by our composition theorem that the 4-round IEM cipher is correlation intractable, and hence offers some form of resistance to CKAs, but we warn that due to the quadratic query complexity of our simulator, the provable guarantee one obtains is not as tight as one might wish.

A NOTE ON KNOWN-KEY ATTACKS. Known-key attacks refer, informally, to the setting where the adversary is given a block cipher E and a random key k , and must exhibit some non-random behavior of the permutation E_k [22]. In order to capture this security property, Andreeva *et al.* [2] have introduced the notion of known-key indifferentiability (KK-indifferentiability), and they have proved that the 1-round Even-Mansour cipher is KK-indifferentiable from an ideal cipher. This might seem surprising at first sight since KKAs seem stronger

⁴ For example, the fact that for any fixed block cipher E , $E_0(0)$ has some fixed, non-random value may be seen as a non-random behavior, yet arguably a harmless one.

Table 1. Summary of provable security results for the iterated Even-Mansour cipher $\text{EM}[n, r, \gamma]$ (with independent inner permutations). The *trivial* key-schedule means that all round keys are equal to the n -bit master key.

Sec. notion	# rounds	Key sched.	Sec. bound	Sim. complexity (query / time)	Ref.
Single-key	$r \geq 1$	independent	$2^{\frac{rn}{r+1}}$	—	[11]
	1	trivial	$2^{\frac{n}{2}}$	—	[15, 16]
	2	trivial	$2^{\frac{2n}{3}}$	—	[10]
XOR	3	trivial	$2^{\frac{n}{2}}$	—	this paper
	1	nonlinear	$2^{\frac{n}{2}}$	—	this paper
Chosen-Key (Seq-indiff.)	4	trivial	$2^{\frac{n}{4}}$	q^2 / q^2	this paper
Full indiff.	5	random oracle	$2^{\frac{n}{10}}$	q^2 / q^3	[1]
	12	trivial	$2^{\frac{n}{12}}$	q^4 / q^6	[24]

than RKAs, yet the 1-round Even-Mansour cipher withstands the former but not the latter. We argue however that this is due to the fact that the KK-indifferentiability notion of [2] is slightly too restrictive because it involves one single random key. We defer the details to Appendix B.

RELATED WORK. Provable security against RKAs was already considered in previous work. However, this was either for weak classes of RKAs (in particular, lacking the completeness property) [4, 25], or for inefficient number-theoretic constructions [3]. Our own results seem to be the first that hold both for a natural class of RKAs and for a practically-oriented construction. For provable security against CKAs, the only previous work we are aware of is [26], which considered the 6-round Feistel construction.

In a concurrent and independent work, Farshim and Procter [17] also analyze the related-key security of the iterated Even-Mansour cipher. One of their main results (Corollary 3) is very similar to Theorem 2 in this paper; their bound is slightly worse than ours, but their analysis is more general and applies to other families of related-key deriving functions than the xor-induced family. They also consider chosen-plaintext (related-key) attacks, whereas we directly consider chosen-plaintext and ciphertext attacks.

OPEN PROBLEMS. Regarding related-key security, it seems natural to conjecture that four rounds and the trivial key-schedule on one hand, or two rounds and a nonlinear key-schedule on the other hand, should deliver a $\mathcal{O}(2^{\frac{2n}{3}})$ -security bound. If true, this should be provable by combining the techniques of [10] and the techniques of this paper. Regarding chosen-key security, an interesting open

problem would be to find a construction of a block cipher from some underlying primitive (e.g., a random oracle or a small set of random permutations) which is seq-indifferentiable from an ideal cipher with a linear simulator complexity (indeed, by our composition theorem, this would imply an optimal resistance to CKAs). A first step in this direction was taken by Kiltz *et al.* [21] in the context of digital signatures.

ORGANIZATION. We set the notation and give some useful definitions in Section 2. We then consider the security of the IEM cipher against RKAs in Section 3 and against CKAs in Section 4. Some proofs are omitted for reasons of space and can be found in the full version of the paper [12].

2 Preliminaries

GENERAL NOTATION. In all the following, we fix an integer $n \geq 1$ and denote $N = 2^n$. The set of all permutations on $\{0, 1\}^n$ will be denoted \mathcal{P}_n . A block cipher with key space $\{0, 1\}^\kappa$ and message space $\{0, 1\}^n$ is a mapping $E : \{0, 1\}^\kappa \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that for any key $k \in \{0, 1\}^\kappa$, $x \mapsto E(k, x)$ is a permutation. We interchangeably use the notations $E(k, x)$ and $E_k(x)$. We denote $\mathbf{BC}(\kappa, n)$ the set of all block ciphers with key space $\{0, 1\}^\kappa$ and message space $\{0, 1\}^n$. For integers $1 \leq s \leq t$, we will write $(t)_s = t(t-1)\cdots(t-s+1)$ and $(t)_0 = 1$ by convention. For a function $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$, let

$$\delta(f) = \max_{a, b \in \{0, 1\}^n, a \neq 0} |\{x \in \{0, 1\}^n : f(x \oplus a) \oplus f(x) = b\}|.$$

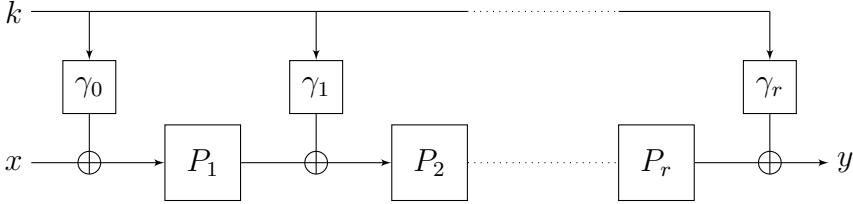
Note that $\delta(f)$ is a measure of the nonlinearity of f . A permutation f of $\{0, 1\}^n$ is said *almost perfect nonlinear* [28] if $\delta(f) = 2$.

THE ITERATED EVEN-MANSOUR CIPHER. Fix integers $n, r \geq 1$. Let $\gamma = (\gamma_0, \dots, \gamma_r)$ be a $(r+1)$ -tuple of permutations of $\{0, 1\}^n$. The r -round iterated Even-Mansour construction $\mathbf{EM}[n, r, \gamma]$ specifies, from any r -tuple $P = (P_1, \dots, P_r)$ of permutations of $\{0, 1\}^n$, a block cipher with n -bit keys and n -bit messages, simply denoted \mathbf{EM}^P in all the following (parameters $[n, r, \gamma]$ will always be clear from the context), which maps a plaintext $x \in \{0, 1\}^n$ and a key $k \in \{0, 1\}^n$ to the ciphertext defined by (see Figure 1):

$$\mathbf{EM}^P(k, x) = \gamma_r(k) \oplus P_r(\gamma_{r-1}(k) \oplus P_{r-1}(\cdots P_2(\gamma_1(k) \oplus P_1(\gamma_0(k) \oplus x)) \cdots)).$$

The pseudorandomness of the IEM cipher was mostly studied for the case of *independent* round keys [8, 11, 23], with the notable exception of [10]. In this paper, we focus on the case where the round keys are derived from an n -bit master key.

RELATED-KEY ORACLE. Let $E \in \mathbf{BC}(\kappa, n)$ be a block cipher, and fix a key $k \in \{0, 1\}^\kappa$. We define the xor-restricted related-key oracle $\mathbf{RK}[E_k]$, which takes as input an “offset” $\Delta \in \{0, 1\}^\kappa$ and a plaintext $x \in \{0, 1\}^n$, and returns $\mathbf{RK}[E_k](\Delta, x) := E_{k \oplus \Delta}(x)$. The oracle can be queried backward, which we denote $\mathbf{RK}[E_k]^{-1}(\Delta, y) := E_{k \oplus \Delta}^{-1}(y)$.

**Fig. 1.** The r -round iterated Even-Mansour cipher

3 Resistance to Related-Key Attacks

3.1 Security Definitions

To formalize related-key attacks against the r -round IEM cipher, we extend in a straightforward way the classical Bellare-Kohno model [4] to the case where the adversary has access to additional oracles. Formally, we consider a xor-restricted related-key adversary \mathcal{D} which has access to $r+1$ oracles, a related-key oracle and r permutation oracles, and must distinguish between the following two worlds:

- the “real” world, where it interacts with $(\text{RK}[\text{EM}_k^P], P)$ where $P = (P_1, \dots, P_r)$ is a tuple of random permutations and k is a randomly drawn key;
- the “ideal” world where it interacts with $(\text{RK}[E_k], P)$ where $P = (P_1, \dots, P_r)$ is a tuple of random permutations, E is an ideal cipher independent from P , and k a randomly drawn key.

The distinguisher is adaptive, and can make two-sided queries to each oracle. As usual, we assume that it is computationally unbounded, deterministic, and never makes pointless queries. Note that in the ideal world, the key k is meaningless, and the related-key oracle $\text{RK}[E_k]$ simply implements an independent random permutation for each offset $\Delta \in \{0, 1\}^n$.

The distinguishing advantage of \mathcal{D} is defined as

$$\mathbf{Adv}(\mathcal{D}) = \left| \Pr \left[\mathcal{D}^{\text{RK}[\text{EM}_k^P], P} = 1 \right] - \Pr \left[\mathcal{D}^{\text{RK}[E_k], P} = 1 \right] \right|,$$

where the first probability is taken over the random choice of k and P , and the second probability is taken over the random choice of E , k , and P .

For q_e, q_p non-negative integers, we define the insecurity of the iterated Even-Mansour cipher against xor-restricted related-key attacks as

$$\mathbf{Adv}_{\text{EM}[n, r, \gamma]}^{\text{xor-rka}}(q_e, q_p) = \max_{\mathcal{D}} \mathbf{Adv}(\mathcal{D}),$$

where the maximum is taken over all distinguishers making exactly q_e queries to the related-key oracle and exactly q_p queries to each inner permutation oracle.

TRANSCRIPT. We summarize the information gathered by the distinguisher in what we call the *query transcript* $(\mathcal{Q}_E, \mathcal{Q}_{P_1}, \dots, \mathcal{Q}_{P_r})$, defined as follows. The tuple

$$\mathcal{Q}_E = ((\Delta_1, x_1, y_1), \dots, (\Delta_{q_e}, x_{q_e}, y_{q_e}))$$

summarizes the queries to the related-key oracle, and means that the j -th query was either a forward query (Δ_j, x_j) and the answer y_j , or a backward query (Δ_j, y_j) and the answer x_j . Similarly, the tuple

$$\mathcal{Q}_{P_i} = ((u_{i,1}, v_{i,1}), \dots, (u_{i,q_p}, v_{i,q_p}))$$

summarizes the queries to the i -th inner permutation P_i , and means that the j -th query was either a forward query $u_{i,j}$ and the answer $v_{i,j}$, or a backward query $v_{i,j}$ and the answer $u_{i,j}$. (Recall that the distinguisher is deterministic, so that there is a one-to-one mapping between this directionless representation and the raw transcript of the interaction of the distinguisher with the oracles). A query transcript is said *attainable* if the probability to obtain it in the ideal world is non-zero (hence, the set of attainable query transcripts depends on the distinguisher). To simplify the security proof (in particular, the definition of *bad* transcripts), we reveal to the distinguisher the key k at the end of its query phase (this is without loss of generality since \mathcal{D} is free to ignore this additional information to compute its output bit). Formally, we append k to the query transcript $(\mathcal{Q}_E, \mathcal{Q}_{P_1}, \dots, \mathcal{Q}_{P_r})$, obtaining what we will simply call the *transcript* $\tau = (\mathcal{Q}_E, \mathcal{Q}_{P_1}, \dots, \mathcal{Q}_{P_r}, k)$ of the attack. A transcript τ is said attainable if the corresponding query transcript is attainable. We denote \mathcal{T} the set of attainable transcripts. In all the following, we denote T_{re} , resp. T_{id} , the probability distribution of the transcript τ induced by the real world, resp. the ideal world (note that these two probability distributions depend on the distinguisher). By extension, we use the same notation to denote a random variable distributed according to each distribution.

ADDITIONAL NOTATION. Given a block cipher $E \in \mathsf{BC}(n, n)$, a key $k \in \{0, 1\}^n$, and a related-key oracle query transcript \mathcal{Q}_E , we say that (E, k) extends \mathcal{Q}_E , written $(E, k) \vdash \mathcal{Q}_E$, if $E_{k \oplus \Delta}(x) = y$ for each $(\Delta, x, y) \in \mathcal{Q}_E$. Similarly, given a permutation P and a permutation query transcript \mathcal{Q}_P , we say that P extends \mathcal{Q}_P , written $P \vdash \mathcal{Q}_P$, if $P(u) = v$ for each $(u, v) \in \mathcal{Q}_P$. It is easy to see that for any attainable transcript $\tau = (\mathcal{Q}_E, \mathcal{Q}_{P_1}, \dots, \mathcal{Q}_{P_r}, k)$, the interaction of the distinguisher with oracles $(\mathsf{RK}[E_k], P_1, \dots, P_r)$ produces τ iff $(E, k) \vdash \mathcal{Q}_E$ and $P_i \vdash \mathcal{Q}_{P_i}$ for $i = 1, \dots, r$.

THE H-COEFFICIENTS TECHNIQUE. We use the H-coefficients technique [29], which relies on the following lemma. See e.g. [10, 11] for a proof.

Lemma 1. *Fix a distinguisher \mathcal{D} . Let $\mathcal{T} = \mathcal{T}_{\text{good}} \sqcup \mathcal{T}_{\text{bad}}$ be a partition of the set of attainable transcripts. Assume that there exists ε_1 such that for any $\tau \in \mathcal{T}_{\text{good}}$, one has⁵*

$$\frac{\Pr[T_{\text{re}} = \tau]}{\Pr[T_{\text{id}} = \tau]} \geq 1 - \varepsilon_1,$$

and that there exists ε_2 such that $\Pr[T_{\text{id}} \in \mathcal{T}_{\text{bad}}] \leq \varepsilon_2$. Then $\mathbf{Adv}(\mathcal{D}) \leq \varepsilon_1 + \varepsilon_2$.

⁵ Recall that for an attainable transcript, one has $\Pr[T_{\text{id}} = \tau] > 0$.

3.2 The Linear Key-Schedule Case

In this section, we consider xor-induced related-key attacks against the IEM cipher with independent permutations and a linear key-schedule. We give attacks for up to two rounds, and then prove a $\mathcal{O}(2^{\frac{n}{2}})$ -security bound for three rounds.

A SIMPLE ATTACK ON ONE ROUND. We start with a very simple attack for one round. Given a permutation P on $\{0, 1\}^n$ and two linear permutations $\gamma_0, \gamma_1 : \{0, 1\}^n \rightarrow \{0, 1\}^n$, consider the 1-round Even-Mansour cipher which maps a key $k \in \{0, 1\}^n$ and a plaintext $x \in \{0, 1\}^n$ to the ciphertext defined as

$$\text{EM}^P(k, x) = \gamma_1(k) \oplus P(\gamma_0(k) \oplus x).$$

Consider the distinguisher which simply queries the related-key oracle on two inputs $(0, x)$ and $(\Delta, x \oplus \gamma_0(\Delta))$, where $\Delta \neq 0$, getting respective answers y and y' , and checks whether $y' = y \oplus \gamma_1(\Delta)$. This holds with probability 1 in the real world, but only with probability $1/N$ in the ideal world, so that the distinguishing advantage of this adversary is negligibly close to one.

AN ATTACK ON TWO ROUNDS. We then show a more intricate distinguishing attack for two rounds (and, again, a linear key-schedule). This attack does not require to query the internal permutation oracles, and makes only four queries to the related-key oracle. It can be seen as a very efficient boomerang related-key attack [5]. Formally, we prove the following theorem.

Theorem 1. *Let $\gamma = (\gamma_0, \gamma_1, \gamma_2)$ be a linear key-schedule. Then*

$$\mathbf{Adv}_{\text{EM}[n, 2, \gamma]}^{\text{xor-rka}}(4, 0) \geq 1 - \frac{1}{N}.$$

Proof. We denote generically $(\text{RK}, (P_1, P_2))$ the oracles to which the adversary has access. Consider the following distinguisher (see Figure 2 for a diagram of the attack):

- (1) choose arbitrary values $x_1, \Delta_1 \in \{0, 1\}^n$, and query $y_1 := \text{RK}(\Delta_1, x_1)$;
- (2) choose an arbitrary value $\Delta_2 \in \{0, 1\}^n \setminus \{\Delta_1\}$, compute $x_2 := x_1 \oplus \gamma_0(\Delta_2 \oplus \Delta_1)$, and query $y_2 := \text{RK}(\Delta_2, x_2)$;
- (3) choose an arbitrary $\Delta_3 \in \{0, 1\}^n \setminus \{\Delta_1, \Delta_2\}$, compute $y_3 := y_1 \oplus \gamma_2(\Delta_1 \oplus \Delta_3)$, and query $x_3 := \text{RK}^{-1}(\Delta_3, y_3)$;
- (4) compute $\Delta_4 := \Delta_3 \oplus \Delta_2 \oplus \Delta_1$ and $y_4 := y_2 \oplus \gamma_2(\Delta_2 \oplus \Delta_4)$, and query $x_4 := \text{RK}^{-1}(\Delta_4, y_4)$;
- (5) if $x_4 = x_3 \oplus \gamma_0(\Delta_3 \oplus \Delta_4)$, output 1, else output 0.

When the distinguisher is interacting with the ideal world $(\text{RK}[E], (P_1, P_2))$, where E is an ideal cipher independent from P_1 and P_2 , the value x_4 is uniformly random and independent from x_3 , Δ_3 , and Δ_4 (indeed the offsets Δ_i for $i = 1, 2, 3, 4$ are pairwise distinct, so that y_4 is the first query to the random permutation corresponding to offset Δ_4). Hence, the probability that the distinguisher returns 1 in the ideal case is 2^{-n} .

Now we show that when the distinguisher is interacting with the real world, i.e., with $(\text{RK}[\text{EM}_k^{P_1, P_2}], (P_1, P_2))$, it always returns 1, independently of k , P_1 , and P_2 . Noting that, by definition, $x_2 = x_1 \oplus \gamma_0(\Delta_2 \oplus \Delta_1)$, we denote u_1 the common value

$$u_1 \stackrel{\text{def}}{=} x_1 \oplus \gamma_0(k \oplus \Delta_1) = x_2 \oplus \gamma_0(k \oplus \Delta_2),$$

and we denote $v_1 = P_1(u_1)$. We also denote

$$u_2 = v_1 \oplus \gamma_1(k \oplus \Delta_1) \tag{1}$$

$$\begin{aligned} v_2 &= P_2(u_2) \\ u'_2 &= v_1 \oplus \gamma_1(k \oplus \Delta_2) \\ v'_2 &= P_2(u'_2). \end{aligned} \tag{2}$$

Hence, one has

$$y_1 = v_2 \oplus \gamma_2(k \oplus \Delta_1) \tag{3}$$

$$y_2 = v'_2 \oplus \gamma_2(k \oplus \Delta_2). \tag{4}$$

Since $y_3 = y_1 \oplus \gamma_2(\Delta_1 \oplus \Delta_3)$, we can see, using (3), that

$$y_3 \oplus \gamma_2(k \oplus \Delta_3) = y_1 \oplus \gamma_2(k \oplus \Delta_1) = v_2.$$

Define

$$\begin{aligned} v'_1 &= u_2 \oplus \gamma_1(k \oplus \Delta_3) \\ u'_1 &= P_1^{-1}(v'_1). \end{aligned} \tag{5}$$

This implies that

$$x_3 = u'_1 \oplus \gamma_0(k \oplus \Delta_3). \tag{6}$$

Since $y_4 = y_2 \oplus \gamma_2(\Delta_2 \oplus \Delta_4)$, we see by (4) that

$$y_4 \oplus \gamma_2(k \oplus \Delta_4) = y_2 \oplus \gamma_2(k \oplus \Delta_2) = v'_2.$$

Moreover, since $\Delta_4 = \Delta_3 \oplus \Delta_2 \oplus \Delta_1$, we have

$$\begin{aligned} u'_2 \oplus \gamma_1(k \oplus \Delta_4) &= u'_2 \oplus \gamma_1(k \oplus \Delta_2) \oplus \gamma_1(\Delta_1 \oplus \Delta_3) \\ &= v_1 \oplus \gamma_1(k \oplus \Delta_1) \oplus \gamma_1(k \oplus \Delta_3) && \text{by (2)} \\ &= u_2 \oplus \gamma_1(k \oplus \Delta_3) && \text{by (1)} \\ &= v'_1 && \text{by (5).} \end{aligned}$$

This finally implies by (6) that

$$x_4 = u'_1 \oplus \gamma_0(k \oplus \Delta_4) = x_3 \oplus \gamma_0(\Delta_3 \oplus \Delta_4),$$

which concludes the proof. \square

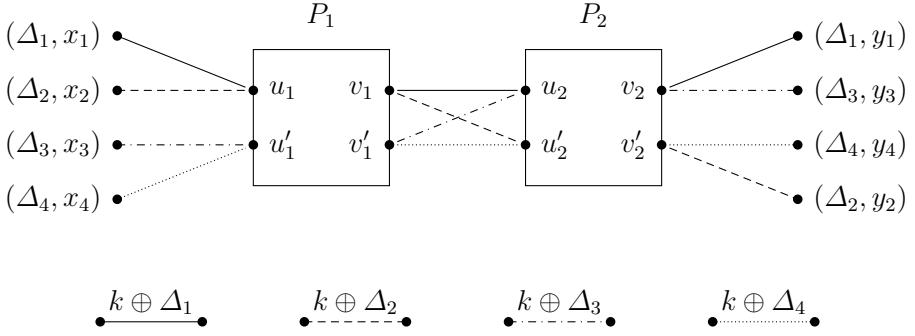


Fig. 2. A related-key attack on the iterated Even-Mansour cipher with two rounds and a linear key-schedule

SECURITY PROOF FOR THREE ROUNDS. We consider the 3-round IEM cipher with the trivial key schedule (the result can be straightforwardly extended to the general case where the key derivation functions $(\gamma_0, \dots, \gamma_3)$ are any permutations). Given three permutations P_1, P_2, P_3 on $\{0, 1\}^n$, we denote $\text{EM}^{P_1, P_2, P_3}$ the 3-round IEM cipher which maps a key $k \in \{0, 1\}^n$ and a plaintext $x \in \{0, 1\}^n$ to the ciphertext defined as

$$\text{EM}^{P_1, P_2, P_3}(k, x) = k \oplus P_3(k \oplus P_2(k \oplus P_1(k \oplus x))).$$

We prove the following result.

Theorem 2. Let q_e, q_p be positive integers, $N = 2^n$, and \mathcal{I} be the trivial key-schedule. Then

$$\text{Adv}_{\text{EM}[n, 3, \mathcal{I}]}^{\text{xor-rka}}(q_e, q_p) \leq \frac{6q_e q_p}{N} + \frac{4q_e^2}{N}.$$

Proof. The proof follows from Lemma 1, and Lemmas 2 and 3 proven below. \square

Following the H-coefficient technique, we start by defining bad transcripts.

Definition 1. Let $\tau = (\mathcal{Q}_E, \mathcal{Q}_{P_1}, \mathcal{Q}_{P_2}, \mathcal{Q}_{P_3}, k)$ be an attainable transcript. We say that τ is bad if

$$k \in \text{BadK} = \bigcup_{1 \leq i \leq 2} \text{BadK}_i$$

where:

$$k \in \text{BadK}_1 \Leftrightarrow \text{there exists } (\Delta, x, y) \in \mathcal{Q}_E \text{ and } (u_1, v_1) \in \mathcal{Q}_{P_1} \text{ such that} \\ k \oplus \Delta = x \oplus u_1$$

$$k \in \text{BadK}_2 \Leftrightarrow \text{there exists } (\Delta, x, y) \in \mathcal{Q}_E \text{ and } (u_3, v_3) \in \mathcal{Q}_{P_3} \text{ such that} \\ k \oplus \Delta = y \oplus v_3.$$

Otherwise, τ is said good. We denote \mathcal{T}_{bad} the set of bad transcripts, and $\mathcal{T}_{\text{good}} = \mathcal{T} \setminus \mathcal{T}_{\text{bad}}$ the set of good transcripts.

First, we upper bound the probability to get a bad transcript in the ideal world.

Lemma 2.

$$\Pr[T_{\text{id}} \in \mathcal{T}_{\text{bad}}] \leq \frac{2q_e q_p}{N}.$$

Proof. Since we are in the ideal case, the key k is drawn uniformly at random at the end of the query phase. Hence, we only need to upper bound the number of possible bad values for k for every attainable query transcripts $(\mathcal{Q}_E, \mathcal{Q}_{P_1}, \mathcal{Q}_{P_2}, \mathcal{Q}_{P_3})$. Fix any query transcript $(\mathcal{Q}_E, \mathcal{Q}_{P_1}, \mathcal{Q}_{P_2}, \mathcal{Q}_{P_3})$. Then, for every $(\Delta, x, y) \in \mathcal{Q}_E$ and every $(u_1, v_1) \in \mathcal{Q}_{P_1}$, there is exactly one key k such that $k = x \oplus \Delta \oplus u_1$. Hence, $|\text{BadK}_1| \leq q_e q_p$. Similarly, $|\text{BadK}_2| \leq q_e q_p$. Hence, for $i = 1, 2$,

$$\Pr[k \leftarrow_{\$} \{0, 1\}^n : k \in \text{BadK}_i] \leq \frac{q_e q_p}{N}.$$

The result follows. \square

We then consider good transcripts in the following lemma.

Lemma 3. *For any good transcript $\tau \in \mathcal{T}_{\text{good}}$, one has*

$$\frac{\Pr[T_{\text{re}} = \tau]}{\Pr[T_{\text{id}} = \tau]} \geq 1 - \frac{4q_e q_p}{N} - \frac{4q_e^2}{N}.$$

Proof. If $\mathcal{T}_{\text{good}} = \emptyset$, there is nothing to prove. Otherwise, fix a good transcript $\tau = (\mathcal{Q}_E, \mathcal{Q}_{P_1}, \mathcal{Q}_{P_2}, \mathcal{Q}_{P_3}, k)$. Let m denote the number of different offsets Δ appearing in \mathcal{Q}_E and q_i the number of queries using the i -th offset (ordering the offsets arbitrarily). Note that $q_e = \sum_{i=1}^m q_i$. In the ideal world, one simply has

$$\begin{aligned} \Pr[T_{\text{id}} = \tau] &= \Pr[k' \leftarrow_{\$} \{0, 1\}^n : k' = k] \times \Pr[P_i \leftarrow_{\$} \mathcal{P}_n : P_i \vdash \mathcal{Q}_{P_i}, i = 1, 2, 3] \\ &\quad \times \Pr[E \leftarrow_{\$} \text{BC}(n, n) : (E, k) \vdash \mathcal{Q}_E] \\ &= \frac{1}{N} \cdot \frac{1}{((N)_{q_p})^3} \cdot \frac{1}{\prod_{i=1}^m (N)_{q_i}}. \end{aligned} \tag{7}$$

Now we have to lower bound the probability

$$\begin{aligned} \Pr[T_{\text{re}} = \tau] &= \frac{1}{N} \times \Pr \left[P_1, P_2, P_3 \leftarrow_{\$} \mathcal{P}_n : \right. \\ &\quad \left. (\text{EM}^{P_1, P_2, P_3}, k) \vdash \mathcal{Q}_E \wedge P_i \vdash \mathcal{Q}_{P_i}, i = 1, 2, 3 \right]. \end{aligned}$$

Let

$$\begin{aligned} U_1 &= \{u_1 \in \{0, 1\}^n : (u_1, v_1) \in \mathcal{Q}_{P_1}\}, \quad V_1 = \{v_1 \in \{0, 1\}^n : (u_1, v_1) \in \mathcal{Q}_{P_1}\}, \\ U_2 &= \{u_2 \in \{0, 1\}^n : (u_2, v_2) \in \mathcal{Q}_{P_2}\}, \quad V_2 = \{v_2 \in \{0, 1\}^n : (u_2, v_2) \in \mathcal{Q}_{P_2}\}, \\ U_3 &= \{u_3 \in \{0, 1\}^n : (u_3, v_3) \in \mathcal{Q}_{P_3}\}, \quad V_3 = \{v_3 \in \{0, 1\}^n : (u_3, v_3) \in \mathcal{Q}_{P_3}\} \end{aligned}$$

denote the domains and ranges of \mathcal{Q}_{P_1} , \mathcal{Q}_{P_2} , and \mathcal{Q}_{P_3} respectively. For $u'_1 \in \{0, 1\}^n$, let $X(u'_1) = \{(\Delta, x, y) \in \mathcal{Q}_E : x \oplus k \oplus \Delta = u'_1\}$, and let $U'_1 = \{u'_1 \in$

$\{0, 1\}^n : X(u'_1) \neq \emptyset\}$. Similarly, for $v'_3 \in \{0, 1\}^n$, let $Y(v'_3) = \{(\Delta, x, y) \in \mathcal{Q}_E : y \oplus k \oplus \Delta = v'_3\}$, and let $V'_3 = \{v'_3 \in \{0, 1\}^n : Y(v'_3) \neq \emptyset\}$. Note that by definition of a good transcript, one has $U_1 \cap U'_1 = \emptyset$ and $V_3 \cap V'_3 = \emptyset$. Let also $\alpha = |U'_1|$ and $\beta = |V'_3|$. For clarity, we denote

$$\begin{aligned} U'_1 &= \{u'_{1,1}, \dots, u'_{1,\alpha}\} \\ V'_3 &= \{v'_{3,1}, \dots, v'_{3,\beta}\} \end{aligned}$$

using an arbitrary order. Note that

$$q_e = \sum_{i=1}^{\alpha} |X(u'_{1,i})| = \sum_{i=1}^{\beta} |Y(v'_{3,i})|. \quad (8)$$

It is now sufficient for our result to lower bound the number of possible tuple of values $(v'_{1,1}, \dots, v'_{1,\alpha})$ and $(u'_{3,1}, \dots, u'_{3,\beta})$ such that, conditioned on $P_1(u'_{1,i}) = v'_{1,i}$ for $1 \leq i \leq \alpha$ and $P_3(u'_{3,j}) = v'_{3,j}$ for $1 \leq j \leq \beta$, the event $E_k^{P_1, P_2, P_3} \vdash \mathcal{Q}_E$ is equivalent to q_e “new” equations on P_2 (i.e., distinct from equations imposed by $P_2 \vdash \mathcal{Q}_{P_2}$). More precisely, let N_1 be the number of tuples of pairwise distinct values $(v'_{1,1}, \dots, v'_{1,\alpha})$ such that, for every $i = 1, \dots, \alpha$:

- (i) $v'_{1,i} \neq v_1$ for every $v_1 \in V_1$,
- (ii) $v'_{1,i} \neq k \oplus \Delta \oplus u_2$ for every $(\Delta, x, y) \in X(u'_{1,i})$, $u_2 \in U_2$,
- (iii) $v'_{1,i} \neq \Delta \oplus v'_{1,j} \oplus \Delta'$ for every $(\Delta, x, y) \in X(u'_{1,i})$, $1 \leq j \leq i-1$, $(\Delta', x', y') \in X(u'_{1,j})$.

Then

$$\begin{aligned} N_1 &\geq \prod_{i=1}^{\alpha} \left(N - q_p - i + 1 - |X(u'_{1,i})|(q_p + \sum_{j=1}^{i-1} |X(u'_{1,j})|) \right) \\ &\geq \prod_{i=1}^{\alpha} (N - q_p - q_e - |X(u'_{1,i})|(q_p + q_e)) \quad \text{by (8).} \end{aligned}$$

Similarly, let N_3 be the number of tuples of pairwise distinct values $(u'_{3,1}, \dots, u'_{3,\beta})$ such that, for every $i = 1, \dots, \beta$:

- (i') $u'_{3,i} \neq u_3$ for every $u_3 \in U_3$,
- (ii') $u'_{3,i} \neq k \oplus \Delta \oplus v_2$ for every $(\Delta, x, y) \in Y(v'_{3,i})$, $v_2 \in V_2$,
- (iii') $u'_{3,i} \neq \Delta \oplus u'_{3,j} \oplus \Delta'$ for every $(\Delta, x, y) \in Y(v'_{3,i})$, $1 \leq j \leq i-1$, $(\Delta', x', y') \in Y(v'_{3,j})$.

Then

$$\begin{aligned} N_3 &\geq \prod_{i=1}^{\beta} \left(N - q_p - i + 1 - |Y(v'_{3,i})|(q_p + \sum_{j=1}^{i-1} |Y(v'_{3,j})|) \right) \\ &\geq \prod_{i=1}^{\beta} (N - q_p - q_e - |Y(v'_{3,i})|(q_p + q_e)) \quad \text{by (8).} \end{aligned}$$

For every possible choice of $(v'_{1,1}, \dots, v'_{1,\alpha})$ and $(u'_{3,1}, \dots, u'_{3,\beta})$ satisfying these conditions, P_1 will be fixed on exactly $q_p + \alpha$ points, P_2 on $q_p + q_e$ points and P_3 on $q_p + \beta$ points. In more details, assume $N_1 \cdot N_3 > 0$, fix any tuples $(v'_{1,1}, \dots, v'_{1,\alpha})$ and $(u'_{3,1}, \dots, u'_{3,\beta})$ satisfying these conditions, and let Ev_1 be the event that $P_1(u'_{1,i}) = v'_{1,i}$ for $1 \leq i \leq \alpha$ and Ev_3 be the event that $P_3(u'_{3,j}) = v'_{3,j}$ for $1 \leq j \leq \beta$. Then by conditions (i) and (i') we have

$$\begin{aligned}\Pr[\text{Ev}_1 \wedge (P_1 \vdash \mathcal{Q}_{P_1})] &= \frac{1}{(N)_{q_p+\alpha}} \\ \Pr[\text{Ev}_3 \wedge (P_3 \vdash \mathcal{Q}_{P_3})] &= \frac{1}{(N)_{q_p+\beta}}.\end{aligned}$$

Fix now P_1 and P_3 satisfying Ev_1 and Ev_3 . For each $(\Delta, x, y) \in \mathcal{Q}_E$, let u'_2 and v'_2 be respectively the corresponding input and output to P_2 for this query, viz., $u'_2 = v'_{1,i} \oplus k \oplus \Delta$ for i such that $x \oplus k \oplus \Delta = u'_{1,i}$, and $v'_2 = u'_{3,j} \oplus k \oplus \Delta$ for j such that $y \oplus k \oplus \Delta = v'_{3,j}$. Then, the q_e values u'_2 are all outside U_2 by condition (ii), and pairwise distinct by condition (iii), and similarly the q_e values v'_2 are all outside V_2 by condition (ii'), and pairwise distinct by condition (iii'). It follows that

$$\begin{aligned}\Pr\left[(\text{EM}^{P_1, P_2, P_3}, k) \vdash \mathcal{Q}_E \wedge (P_2 \vdash \mathcal{Q}_{P_2}) \mid \text{Ev}_1 \wedge (P_1 \vdash \mathcal{Q}_{P_1}) \wedge \text{Ev}_3 \wedge (P_3 \vdash \mathcal{Q}_{P_3})\right] \\ = \frac{1}{(N)_{q_p+q_e}}.\end{aligned}$$

Hence, summing over the at least $N_1 \cdot N_3$ possible pairs of tuples, we obtain

$$\Pr[T_{\text{re}} = \tau] \geq \frac{N_1 \cdot N_3}{N \cdot (N)_{q_p+\alpha} \cdot (N)_{q_p+q_e} \cdot (N)_{q_p+\beta}}. \quad (9)$$

This last inequality is also trivially true if $N_1 \cdot N_3 = 0$. Using (7) and (9), one has

$$\begin{aligned}\frac{\Pr[T_{\text{re}} = \tau]}{\Pr[T_{\text{id}} = \tau]} &\geq \frac{N_1 \cdot N_3 \cdot N \cdot (N)_{q_p}^3 \prod_{i=1}^m (N)_{q_i}}{N \cdot (N)_{q_p+\alpha} \cdot (N)_{q_p+q_e} \cdot (N)_{q_p+\beta}} \\ &\geq \frac{N_1 \cdot N_3 \cdot \prod_{i=1}^m (N)_{q_i}}{(N - q_p)_\alpha \cdot (N - q_p)_{q_e} \cdot (N - q_p)_\beta} \\ &\geq \frac{N_1 \cdot N_3 \cdot (N)_{q_e}}{(N - q_p)_\alpha \cdot (N - q_p)_{q_e} \cdot (N - q_p)_\beta} \\ &\geq \frac{N_1 \cdot N_3}{N^{\alpha+\beta}}.\end{aligned}$$

Finally, one has, since $\alpha \leq q_e$,

$$\begin{aligned} \frac{N_1}{N^\alpha} &= \frac{\prod_{i=1}^\alpha (N - q_p - q_e - |X(u'_{1,i})|(q_p + q_e))}{N^\alpha} \\ &\geq 1 - \sum_{i=1}^\alpha \frac{q_p + q_e + |X(u'_{1,i})|(q_p + q_e)}{N} \\ &\geq 1 - \frac{q_e q_p}{N} - \frac{q_e^2}{N} - (q_p + q_e) \sum_{i=1}^\alpha \frac{|X(u'_{1,i})|}{N} \\ &\geq 1 - \frac{2q_e q_p}{N} - \frac{2q_e^2}{N} \end{aligned} \quad \text{by (8).}$$

The same lower bound holds for $\frac{N_3}{N^\beta}$. Hence

$$\begin{aligned} \frac{\Pr[T_{\text{re}} = \tau]}{\Pr[T_{\text{id}} = \tau]} &\geq \left(1 - \frac{2q_e q_p}{N} - \frac{2q_e^2}{N}\right)^2 \\ &\geq 1 - \frac{4q_e q_p}{N} - \frac{4q_e^2}{N}. \end{aligned}$$

□

3.3 The Nonlinear Key-Schedule Case

In this section, we show that when the key-schedule is nonlinear, one round is sufficient to achieve a $\mathcal{O}(2^{\frac{n}{2}})$ -security bound against xor-induced related-key attacks.

Given a permutation P on $\{0, 1\}^n$ and two permutations $\gamma_0, \gamma_1 : \{0, 1\}^n \rightarrow \{0, 1\}^n$, we denote EM^P the 1-round Even-Mansour cipher which maps a key $k \in \{0, 1\}^n$ and a plaintext $x \in \{0, 1\}^n$ to the ciphertext defined as

$$\text{EM}^P(k, x) = \gamma_1(k) \oplus P(\gamma_0(k) \oplus x).$$

We prove the following result.

Theorem 3. *Let q_e, q_p be positive integers, $N = 2^n$, and $\gamma = (\gamma_0, \gamma_1)$. Then*

$$\mathbf{Adv}_{\text{EM}[n, 1, \gamma]}^{\text{xor-rka}}(q_e, q_p) \leq \frac{2q_e q_p}{N} + \frac{(\delta(\gamma_0) + \delta(\gamma_1))q_e^2}{2N}.$$

In particular, if γ_0 and γ_1 are almost perfect nonlinear permutations, then

$$\mathbf{Adv}_{\text{EM}[n, 1, \gamma]}^{\text{xor-rka}}(q_e, q_p) \leq \frac{2q_e q_p + 2q_e^2}{N}.$$

Proof. Deferred to the full version of the paper [12] for reasons of space. □

4 Resistance to Chosen-Key Attacks and Sequential Indifferentiability

4.1 Formalizing Chosen-Key Attacks in Idealized Models

In this section, we see a block cipher $E \in \text{BC}(\kappa, n)$ as a primitive which takes as input a triple $\alpha = (\delta, k, z)$, where $\delta \in \{+, -\}$ indicates whether this is a direct (plaintext) or inverse (ciphertext) query, $k \in \{0, 1\}^\kappa$ is the key, and $z \in \{0, 1\}^n$ is the plaintext/ciphertext (depending on δ), and returns the corresponding ciphertext/plaintext (again, depending on δ) $z' \in \{0, 1\}^n$. This allows the block cipher to be described as having a single interface rather than two interfaces E and E^{-1} . In the following, we denote $\text{Dom} = \{+, -\} \times \{0, 1\}^\kappa \times \{0, 1\}^n$ and $\text{Rng} = \{0, 1\}^n$ respectively the domain and the range of E . For an integer $m \geq 1$, an m -ary relation \mathcal{R} is simply a subset $\mathcal{R} \subset \text{Dom}^m \times \text{Rng}^m$.

It is well-known that it is impossible to rigorously define a notion of resistance to chosen-key attacks for block ciphers in the standard model (i.e., for block ciphers not relying on an underlying ideal primitive) without running into impossibility results similar to the one of [9] about random oracles. However, it is possible to avoid such pitfalls in idealized models, as we explain now.

For this, we introduce the concept of evasive relation which, informally, refers to a relation such that it is hard for an algorithm with oracle access to an ideal cipher E to come with a tuple of inputs $(\alpha_1, \dots, \alpha_m)$ such that $((\alpha_1, \dots, \alpha_m), (E(\alpha_1), \dots, E(\alpha_m)))$ satisfies this relation.

Definition 2 (Evasive Relation). An m -ary relation \mathcal{R} is said (q, ε) -evasive (with respect to an ideal cipher) if for any oracle Turing machine \mathcal{M} making at most q oracle queries, one has

$$\Pr \left[E \leftarrow_{\$} \text{BC}(\kappa, n), (\alpha_1, \dots, \alpha_m) \leftarrow \mathcal{M}^E : ((\alpha_1, \dots, \alpha_m), (E(\alpha_1), \dots, E(\alpha_m))) \in \mathcal{R} \right] \leq \varepsilon,$$

where the probability is taken over the random draw of E and the random coins of \mathcal{M} .

Example 1. Consider the problem of finding a preimage of zero for a compression function $f(k, x) := E(k, x) \oplus x$ built from a block cipher E in Davies-Meyer mode, i.e., finding a pair (k, x) such that $E(k, x) \oplus x = 0$. This corresponds to the unary relation $\mathcal{R} = \{((+, k, x), y) \in \text{Dom} \times \text{Rng} : x \oplus y = 0\}$. A celebrated result by Winternitz [33], generalized by Black *et al.* [7], says that this relation is $(q, \mathcal{O}(q/2^n))$ -evasive with respect to an ideal cipher. Similarly, the collision resistance of the Davies-Meyer mode [7] can be recast as a binary $(q, \mathcal{O}(q^2/2^n))$ -evasive relation for the underlying block cipher.

Definition 3 (Correlation Intractable Block Cipher). Let \mathcal{C} be a block cipher construction using (in a black-box way) an underlying primitive F , and let \mathcal{R} be an m -ary relation. \mathcal{C}^F is said to be (q, ε) -correlation intractable with respect to \mathcal{R} if for any oracle Turing machine \mathcal{M} making at most q oracle queries, one has

$$\Pr [(\alpha_1, \dots, \alpha_m) \leftarrow \mathcal{M}^F : ((\alpha_1, \dots, \alpha_m), (\mathcal{C}^F(\alpha_1), \dots, \mathcal{C}^F(\alpha_m))) \in \mathcal{R}] \leq \varepsilon,$$

where the probability is taken over the random draw of F (in some well-understood set) and the random coins of \mathcal{M} .

Informally, a block cipher construction \mathcal{C}^F can be deemed resistant to chosen-key attacks if for any (q, ε) -evasive relation \mathcal{R} , \mathcal{C}^F is (q', ε') -correlation intractable with respect to \mathcal{R} with $q' \simeq q$ and $\varepsilon' \simeq \varepsilon$. Note that our definitions above are information-theoretic, since later we will be able to prove information-theoretic security for the 4-round IEM cipher. There is no obstacle in providing corresponding computational definitions by taking the running time of the algorithms into account.

4.2 Sequential Indifferentiability

We define here the notion of *sequential indifferentiability* (*seq-indifferentiability* for short), introduced by [26], which is a weakened variant of (full) indifferentiability as introduced by [27], and then explain how it is related to correlation intractability. We use the definition of sequential indifferentiability given in [26], tailored to the case of block ciphers.

We start with some definitions. Let \mathcal{C} be a block cipher construction using in a black-box way an underlying primitive F . Let \mathcal{D} be a distinguisher accessing a pair of oracles that we denote generically (E, F) , which can be either the construction together with the underlying primitive F , i.e., (\mathcal{C}^F, F) , or (E, \mathcal{S}^E) where E is an ideal cipher and \mathcal{S} is an oracle Turing machine with oracle access to E called a *simulator*. We will refer informally to E as the *left oracle* and F as the *right oracle*. A distinguisher is said to be *sequential* if after its first query to its left (construction/ideal cipher) oracle, it does not query its right (primitive/simulator) oracle any more. Hence, such a distinguisher works in two phases: first it queries only its right oracle, and then only its left oracle (see Figure 3). We define the *total oracle query cost* of \mathcal{D} as the total number of queries received by F (from \mathcal{D} or \mathcal{C}) when \mathcal{D} interacts with (\mathcal{C}^F, F) . In particular, if \mathcal{C} makes c queries to F to answer any query it receives, and if \mathcal{D} makes q_e queries to its left oracle and q_f queries to its right oracle, then the total oracle query cost of \mathcal{D} is at most $q_f + cq_e$.

Definition 4 (Seq-indifferentiability). Let $q, \sigma, t \in \mathbb{N}$ and $\varepsilon \in \mathbb{R}^+$. A block cipher construction \mathcal{C} with black-box access to an ideal primitive F is said to be $(q, \sigma, t, \varepsilon)$ -seq-indifferentiable from an ideal cipher if there exists an oracle algorithm \mathcal{S} such that for any sequential distinguisher \mathcal{D} of total oracle query

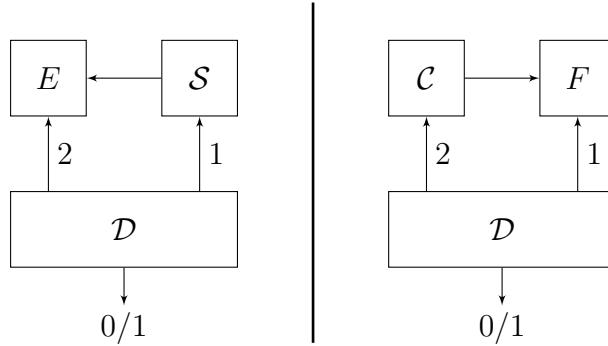


Fig. 3. The sequential indifferentiability notion. The numbers next to query arrows indicate in which order the distinguisher accesses both oracles. After its first query to the left oracle, the distinguisher cannot query the right oracle any more.

cost at most q , \mathcal{S} makes at most σ oracle queries, runs in time at most t , and one has

$$\left| \Pr[\mathcal{D}^{E, \mathcal{S}^E} = 1] - \Pr[\mathcal{D}^{C^F, F} = 1] \right| \leq \varepsilon,$$

where the first probability is taken over the random draw of the ideal cipher E and the random coins of \mathcal{S} , and the second probability is taken over the random draw of F (from some well understood set).

Note that this definition is information-theoretic (the distinguisher might be computationally unbounded), and demands the existence of a *universal* simulator (this is sometimes called *strong* indifferentiability; when the simulator is allowed to depend on the distinguisher, this is called *weak* indifferentiability).

The usefulness of seq-indifferentiability in the context of CKAs comes from the following theorem (the proof is essentially similar to the proof of [26, Theorem 3], but we make the relation between the various parameters explicit).

Theorem 4. Let \mathcal{C} be a block cipher construction using (in a black-box way) an underlying primitive F such that \mathcal{C} makes at most c queries to F on any input. Assume that \mathcal{C}^F is $(q+cm, \sigma, t, \varepsilon)$ -seq-indifferentiable from an ideal cipher. Then for any m -ary relation \mathcal{R} , if \mathcal{R} is $(\sigma+m, \varepsilon_{\mathcal{R}})$ -evasive with respect to an ideal cipher, then \mathcal{C}^F is $(q, \varepsilon + \varepsilon_{\mathcal{R}})$ -correlation intractable with respect to \mathcal{R} .

Proof. Assume that there exists an m -ary relation \mathcal{R} which is $(\sigma+m, \varepsilon_{\mathcal{R}})$ -evasive but such that \mathcal{C}^F is not $(q, \varepsilon + \varepsilon_{\mathcal{R}})$ -correlation intractable with respect to \mathcal{R} . Then there exists an oracle machine \mathcal{M} making at most q oracle queries such that \mathcal{M}^F outputs with probability $\varepsilon' > \varepsilon_{\mathcal{R}} + \varepsilon$ a sequence $(\alpha_1, \dots, \alpha_m)$ such that

$$((\alpha_1, \dots, \alpha_m), (\mathcal{C}^F(\alpha_1), \dots, \mathcal{C}^F(\alpha_m))) \in \mathcal{R}.$$

Consider the following sequential distinguisher \mathcal{D} accessing a pair of oracles (E, F) : it runs \mathcal{M} , answering \mathcal{M} 's oracle queries with its own oracle F , until

\mathcal{M} returns a tuple $(\alpha_1, \dots, \alpha_m)$. \mathcal{D} then makes oracle queries $E(\alpha_1), \dots, E(\alpha_m)$ and checks⁶ whether

$$((\alpha_1, \dots, \alpha_m), (E(\alpha_1), \dots, E(\alpha_m))) \in \mathcal{R}.$$

If this is the case it returns 1, otherwise it returns 0. Note that the total oracle query cost of \mathcal{D} is at most $q + cm$.

When the distinguisher is interacting with (\mathcal{C}^F, F) , the probability that it returns 1 is exactly $\varepsilon' > \varepsilon_{\mathcal{R}} + \varepsilon$. On the other hand, when it interacts with (E, \mathcal{S}^E) , then the union of \mathcal{D} and \mathcal{S} is an oracle machine with oracle access to E making at most $\sigma + m$ oracle queries, so that, by definition of a $(\sigma + m, \varepsilon_{\mathcal{R}})$ -evasive relation, \mathcal{D} outputs 1 with probability at most $\varepsilon_{\mathcal{R}}$. Hence, the advantage of the distinguisher is $\varepsilon' - \varepsilon_{\mathcal{R}} > \varepsilon$, which contradicts the $(q + cm, \sigma, \varepsilon)$ -seq-indifferentiability of \mathcal{C} . \square

INTERPRETATION. Assuming c and m are constants which are negligible compared with q and σ , Theorem 4 can be paraphrased as follows: if \mathcal{C} is $(q, \sigma, t, \varepsilon)$ -seq-indifferentiable from an ideal cipher, and if a relation \mathcal{R} cannot be found with probability better than $\varepsilon_{\mathcal{R}}$ with σ queries to an ideal cipher, then \mathcal{R} cannot be found for \mathcal{C}^F with probability better than $\varepsilon + \varepsilon_{\mathcal{R}}$ with q queries to F . (Note that the running time of the simulator is irrelevant here since we used an information-theoretic definition of correlation intractability.) Hence, seq-indifferentiability measures how much easier it is to find some relation \mathcal{R} for a block cipher construction \mathcal{C}^F than for an ideal cipher. In a sense, Theorem 4 can be seen as the analogue in the case of sequential indifferentiability of the composition theorem of [27, 30] for full indifferentiability.

If one is only concerned with asymptotic security, then seq-indifferentiability implies correlation intractability in the following sense. Let $(\mathcal{C}_n^F)_{n \in \mathbb{N}}$ be a block cipher construction family indexed by a security parameter n . We simply say that \mathcal{C}_n^F is seq-indifferentiable from an ideal cipher if for any $q \in \text{poly}(n)$, \mathcal{C}_n^F is $(q, \sigma, t, \varepsilon)$ -seq-indifferentiable from an ideal cipher with $\sigma, t \in \text{poly}(n)$ and $\varepsilon \in \text{negl}(n)$. We simply say that \mathcal{C}_n^F is correlation intractable if for any (q, ε) -evasive relation \mathcal{R} (with respect to an ideal cipher) where $q \in \text{poly}(n)$ and $\varepsilon \in \text{negl}(n)$, \mathcal{C}_n^F is (q', ε') -correlation intractable with respect to \mathcal{R} for some $q' \in \text{poly}(n)$ and $\varepsilon' \in \text{negl}(n)$. Then a direct corollary of Theorem 4 is that if \mathcal{C}_n^F is (asymptotically) seq-indifferentiable from an ideal cipher, then it is also (asymptotically) correlation intractable.

However, if we adopt the “concrete” security viewpoint, then the exact seq-indifferentiability parameters are important to quantify how well exactly the construction withstands chosen-key attacks. Consider Example 1 of preimage resistance of the Davies-Meyer compression function, which can be phrased as a $(q, \mathcal{O}(q/2^n))$ -evasive relation \mathcal{R} for the underlying (ideal) cipher. Assume that a block cipher construction \mathcal{C}^F is $(q, \sigma, t, \varepsilon)$ -seq-indifferentiable from an ideal cipher

⁶ Note that we are working in the information-theoretic framework, so that the running time of \mathcal{D} is irrelevant. In the computational framework, one should take into account the time necessary to recognize relation \mathcal{R} .

with, e.g., $\sigma = \mathcal{O}(q^2)$ and $\varepsilon = \mathcal{O}(q^2/2^n)$. Then Theorem 4 implies that \mathcal{C}^F is $(q, \mathcal{O}(q^2/2^n))$ -correlation intractable with respect to \mathcal{R} , or in other words, that the Davies-Meyer compression function based on \mathcal{C}^F is $(q, \mathcal{O}(q^2/2^n))$ -preimage resistant (in the ideal- F model). Hence, the quadratic query complexity of the simulator implies a security loss for correlation intractability. This motivates to look for block cipher constructions that are $(q, \sigma, t, \varepsilon)$ -seq-indifferentiable from an ideal cipher with $\sigma = \mathcal{O}(q)$ and $\varepsilon = \mathcal{O}(q/2^n)$, which we leave for future work.

4.3 Proof of Sequential Indifferentiability for Four Rounds

FOUR ROUNDS ARE NECESSARY. We first recall that Lampe and Seurin gave an attack against full indifferentiability of the 3-round IEM cipher [24] (a different attack has been independently described by Andreeva *et al.* [1]). A closer look at their attack shows that their distinguisher is in fact sequential (we refer to [24] for a detailed description of the attack for reasons of space), so that the 3-round IEM cipher cannot even be seq-indifferentiable from an ideal cipher. Hence, at least four rounds are necessary (and, as we will see now, sufficient) to achieve seq-indifferentiability from an ideal cipher.

MAIN RESULT. We now state and prove the main result of this section regarding the seq-indifferentiability of the 4-round IEM cipher. The proof essentially follows the same lines as the proof of full indifferentiability of [24] for twelve rounds, but is quite simpler since the simulator does not recurse when completing chains.

Theorem 5. *Let $N = 2^n$. For any integer q such that $q^2 \leq N/4$, the 4-round IEM construction (with independent permutations and identical round keys) is $(q, \sigma, t, \varepsilon)$ -seq-indifferentiable from an ideal cipher with n -bit blocks and n -bit keys, with*

$$\sigma = q^2, \quad t = \mathcal{O}(q^2), \quad \text{and} \quad \varepsilon = \frac{68q^4}{N}.$$

Remark 1. It was shown in [26] that for stateless ideal primitives (i.e., primitives whose answers do not depend on the order of the queries it receives), seq-indifferentiability implies public indifferentiability [14, 34], a variant of indifferentiability where the simulator gets to know all queries of the distinguisher to E . Since an ideal cipher is stateless, Theorem 5 implies that the 4-round IEM construction is also publicly indifferentiable from an ideal cipher.

In order to prove this theorem, we will first define a simulator \mathcal{S} , then prove that it runs in polynomial time and makes a polynomial number of queries (Lemma 4), and finally prove that the two systems $\Sigma_1 = (E, \mathcal{S}^E)$ and $\Sigma_3 = (\text{EM}^P, P)$ are indistinguishable, using an intermediate system Σ_2 that we will describe later (Lemmas 6 and 7).

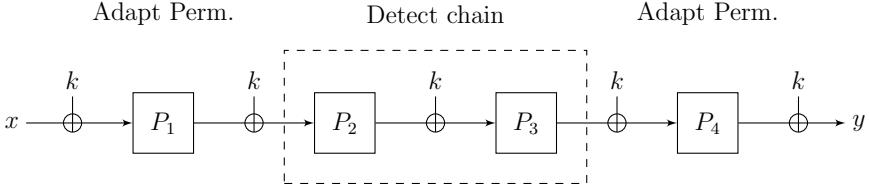


Fig. 4. The 4-round iterated Even-Mansour cipher with independent permutations and identical round keys. The detection and adaptations zones used by the simulator for proving seq-indifferentiability from an ideal cipher are also depicted.

INFORMAL DESCRIPTION OF THE SIMULATOR AND NOTATION. We start with an informal description of the simulator (a formal description in pseudocode is given in Appendix A). The simulator offers an interface $\text{Query}(i, \delta, w)$ to the distinguisher for querying the internal permutations, where $i \in \{1, \dots, 4\}$ names the permutation, $\delta \in \{+, -\}$ indicates whether this is a direct or inverse query, and $w \in \{0, 1\}^n$ is the actual value queried. For each $i = 1, \dots, 4$, the simulator internally maintains a table Π_i mapping entries $(\delta, w) \in \{+, -\} \times \{0, 1\}^n$ to values $w' \in \{0, 1\}^n$, initially undefined for all entries. We denote Π_i^+ , resp. Π_i^- , the (time-dependent) sets of strings $w \in \{0, 1\}^n$ such that $\Pi_i(+, w)$, resp. $\Pi_i(-, w)$, is defined. When the simulator receives a query (i, δ, w) , it looks in table Π_i to see whether the corresponding answer $\Pi_i(\delta, w)$ is already defined. When this is the case, it outputs the answer and waits for the next query. Otherwise, it randomly draws an answer $w' \in \{0, 1\}^n$ and defines $\Pi_i(\delta, w) := w'$ as well as the answer to the opposite query $\Pi_i(\bar{\delta}, w') := w$. In order to handily describe how the answer w' is drawn, we make the randomness used by the simulator explicit through a tuple of random permutations $P = (P_1, \dots, P_4)$. As for the ideal cipher E , we formally let each P_i have a single interface, namely $P_i := \{+, -\} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, and for any $u, v \in \{0, 1\}^n$, $P_i(+, u) = v \Leftrightarrow P_i(-, v) = u$. We assume that the tuple (P_1, \dots, P_4) is drawn uniformly at random at the beginning of the experiment, but we note that \mathcal{S} could equivalently lazily sample these permutations throughout its execution. Then w' is simply defined by the simulator as $w' := P_i(\delta, w)$. (For reasons that will become clear later, this is not equivalent to drawing w' uniformly from $\{0, 1\}^n \setminus \Pi_i^\delta$, see Remark 2.)

After this random choice of the answer w' , and before returning it to the distinguisher, the simulator takes additional steps to ensure consistency with the ideal cipher E by running a *chain completion* mechanism. Namely, if the distinguisher called $\text{Query}(i, \delta, w)$ with $i = 2$ or 3 , the simulator completes all newly created “chains” (v_2, u_3) , where $v_2 \in \Pi_2^-$ and $u_3 \in \Pi_3^+$ by executing a procedure $\text{CompleteChain}(v_2, u_3, \ell)$, where ℓ indicates where the chain will be “adapted”. For example, assume that the distinguisher called $\text{Query}(2, +, u_2)$ and that the answer randomly chosen by the simulator was v_2 (or the backward counterpart, namely the distinguisher called $\text{Query}(2, -, v_2)$ and the answer randomly chosen by the simulator was u_2). Then for each $u_3 \in \Pi_3^+$, the simulator

computes the corresponding key $k := v_2 \oplus u_3$, and evaluates the IEM construction backward, letting $u_2 := \Pi_2(-, v_2)$ and $v_1 := u_2 \oplus k$, and forward, letting $v_3 := \Pi_3(+, u_3)$, $u_4 := v_3 \oplus k$, $v_4 := \Pi_4(+, u_4)$ (setting this value at random in case it was not in Π_4), $y := v_4 \oplus k$, $x := E(-, k, y)$ (hence making a query to E to “wrap around”), and $u_1 := x \oplus k$, until the corresponding input/output values (u_1, v_1) for the first permutation are defined. It then “adapts” (rather than setting randomly) table Π_1 by calling procedure $\text{ForceVal}(u_1, v_1, 1)$ which sets $\Pi_1(+, u_1) := v_1$ and $\Pi_1(-, v_1) := u_1$ in order to ensure consistency of the simulated IEM construction with E . (A crucial point of the proof will be to show that this does not cause an overwrite, i.e., that these two values are undefined before the adaptation occurs.) In case the query was to $\text{Query}(3, \cdot, \cdot)$, the behavior of the simulator is symmetric, namely adaptation of the chain takes place in table Π_4 .

In all the following, we define the *size* of each table Π_i as

$$|\Pi_i| = \max\{|\Pi_i^+|, |\Pi_i^-|\}.$$

(Note that as long as no value is overwritten in the tables, $|\Pi_i^+| = |\Pi_i^-|$.)

Remark 2. As already noted, we could have easily described an equivalent simulator that lazily samples the random permutations (P_1, \dots, P_4) throughout its execution. However, we remark that this is not equivalent to replacing line (6) of the formal description of the simulator in Appendix A by $w' \leftarrow_{\$} \{0, 1\}^n \setminus \Pi_i^\delta$ for $i = 1$ and $i = 4$ since the simulator sometimes adapts the value of these tables, so that the tables Π_i and the permutations P_i will differ in general on the adapted entries.

COMPLEXITY OF THE SIMULATOR. We start by proving that the simulator runs in polynomial time and makes a polynomial number of queries to the ideal cipher. More precisely, we have the following lemma.

Lemma 4. *Consider an execution of the simulator S^E where the simulator receives at most q queries in total. Then:*

- (i) *the size of Π_2 and Π_3 is at most q , and the size of Π_1 and Π_4 is at most $q^2 + q$;*
- (ii) *the simulator executes `CompleteChain` at most q^2 times, makes at most q^2 queries to E , and runs in time $\mathcal{O}(q^2)$.*

Proof. The size of Π_2 , resp. Π_3 , can only increase by one when the distinguisher makes a direct call to $\text{Query}(2, \delta, w)$, resp. $\text{Query}(3, \delta, w)$, so that the size of Π_2 and Π_3 is at most q . Procedure `CompleteChain` is called once for each pair $(v_2, u_3) \in \Pi_2^- \times \Pi_3^+$, hence at most q^2 times in total. Since the simulator makes exactly one query to E per execution of `CompleteChain`, the total number of queries made by the simulator to E is at most q^2 . The size of Π_1 , resp. Π_4 , can only increase by one when the distinguisher calls $\text{Query}(1, \delta, w)$, resp. $\text{Query}(4, \delta, w)$, or when `CompleteChain` is called, hence the size of Π_1 and Π_4 is at most $q^2 + q$. Clearly, the simulator running time is dominated by the executions of `CompleteChain`, hence the simulator runs in time $\mathcal{O}(q^2)$. \square

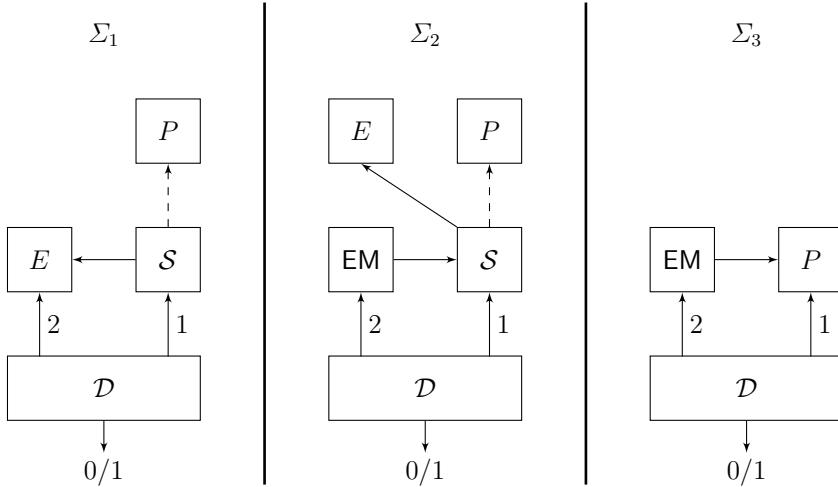


Fig. 5. Systems used in the seq-indifferentiability proof

INTERMEDIATE SYSTEM. In all the following, we consider some fixed distinguisher \mathcal{D} , and assume that it is deterministic (this is *wlog* since we consider computationally unbounded distinguishers). We will denote $\mathcal{S}(E, P)$ rather than $\mathcal{S}(P)^E$ the simulator with oracle access to the ideal cipher E and using random permutations P as source of randomness. In order to prove the indistinguishability of the two systems $(E, \mathcal{S}(E, P))$ and (EM^P, P) , we will use an intermediate system.⁷ Let Σ_1 be the “ideal” world where the distinguisher interacts with $(E, \mathcal{S}(E, P))$. Note that all the randomness of system Σ_1 is captured by the pair (E, P) . Let also Σ_3 be the “real” world where the distinguisher interacts with (EM^P, P) . All the randomness of system Σ_3 is captured by P . In the intermediate system Σ_2 , the distinguisher interacts with $(\text{EM}^{\mathcal{S}(E, P)}, \mathcal{S}(E, P))$ (see Figure 5). In words, the right oracle is the simulator $\mathcal{S}(E, P)$ with oracle access to an ideal cipher E as in Σ_1 , but now the left oracle is the 4-round IEM construction with oracle access to $\mathcal{S}(E, P)$ (rather than random permutations). As for Σ_1 , all the randomness of system Σ_2 is captured by (E, P) .

TRANSITION FROM Σ_1 TO Σ_2 AND GOOD EXECUTIONS. We first consider the transition from the first to the second system.

Definition 5. A pair (E, P) is said *good* if the simulator never overwrites an entry of its tables Π_i during an execution of $\mathcal{D}^{\Sigma_2(E, P)}$. Otherwise the pair is said *bad*.

⁷ We warn that this intermediate system is different from the one used in [24] to prove full indifferentiability of the 12-round IEM cipher, namely $(\text{EM}^P, \mathcal{S}(\text{EM}^P, P))$. It is in fact analogue to the one used by [26] to prove the seq-indifferentiability of the 6-round Feistel construction.

An overwrite may happen either during a random assignment (line (8) of the formal description of the simulator in Appendix A), or when adapting a chain (lines (48) and (49)). Note that whether a pair (E, P) is good or not depends on the distinguisher \mathcal{D} . We first upper bound the probability that a random pair (E, P) is bad.

Lemma 5. *Consider a distinguisher \mathcal{D} of total oracle query cost at most q , with $q^2 \leq N/4$. Then a uniformly random pair (E, P) , where $E \leftarrow_{\$} \text{BC}(n, n)$ and $P \leftarrow_{\$} (\mathcal{P}_n)^4$, is bad (with respect to \mathcal{D}) with probability at most $\frac{16q^4}{N}$.*

Proof. First, note that the total number of queries received by the simulator in Σ_2 (either from \mathcal{D} or from the construction **EM**) is exactly the total oracle query cost q of the distinguisher. Since entries in Π_2 and Π_3 are never adapted, they can never be overwritten either. Hence, we only need to consider the probability of an overwrite in Π_1 or Π_4 . Let **BadRand** be the event that an overwrite occurs during a random assignment (i.e., at line (8)) and **BadAdapt** be the event that an overwrite occurs when adapting a chain (v_2, u_3) (i.e., at line (48) or (49)).

We first consider the probability of **BadRand**. Consider a random assignment in Π_i , for $i = 1$ or 4 , namely $\Pi_i(\delta, w) := w'$, $\Pi_i(\bar{\delta}, w') := w$, with w' randomly defined as $w' := P_i(\delta, w)$. By Lemma 4 (i), there are at most $q^2 + q$ random assignments in Π_1 and Π_4 , so that w' is uniformly random in a set of size at least $N - (q^2 + q)$. Moreover, this random assignment cannot overwrite a value that was previously added during a random assignment, but only a value that was added by **ForceVal** (i.e., when adapting a chain), and by Lemma 4 (ii) there are at most q^2 such values. Hence, the probability that w' is equal to one of the at most q^2 values previously added in table Π_i by a call to **ForceVal** is at most $\frac{q^2}{N - q^2 - q}$. Summing over the at most $q^2 + q$ random assignments in Π_1 and Π_4 , we get

$$\Pr[\text{BadRand}] \leq 2(q^2 + q) \times \frac{q^2}{N - q^2 - q} \leq \frac{8q^4}{N}. \quad (10)$$

We now consider the probability of **BadAdapt**, conditioned on **BadRand** not happening. Let **BadAdapt** _{i} be the event that a value is overwritten by the i -th call to **ForceVal**. We will upper bound the probability

$$\Pr[\text{BadAdapt}_i | \neg \text{BadRand} \wedge \neg \text{BadAdapt}_j, j = 1, \dots, i-1].$$

Consider the i -th execution of **CompleteChain** (v_2, u_3, ℓ) , and assume that event **BadRand** does not occur and **BadAdapt** _{j} does not occur for $1 \leq j \leq i-1$. This means that no value was overwritten before this i -th call to **CompleteChain**. For concreteness, suppose that this chain completion was triggered by a call to **Query** $(2, \cdot, \cdot)$ from the distinguisher, so that $\ell = 1$ (the reasoning is symmetric for a call to **Query** $(3, \cdot, \cdot)$ for which $\ell = 4$). The simulator will eventually call **ForceVal** $(u_1, v_1, 1)$, and we must show that with high probability, the values $\Pi_1(+, v_1)$ and $\Pi_1(-, v_1)$ are undefined previously to this call. We first consider the case of v_1 . This value is defined by the simulator by setting $k := v_2 \oplus u_3$ and $v_1 := u_2 \oplus k$, hence $v_1 = u_2 \oplus v_2 \oplus u_3$. Independently of the direction of the

query of the distinguisher, and since there are at most q random assignments in Π_2 , the value $u_2 \oplus v_2$ comes at random from a set of size at least $N - q$ (if the distinguisher called $\text{Query}(2, +, u_2)$ then v_2 is random, whereas if it called $\text{Query}(2, -, v_2)$ then u_2 is random). Hence, the probability that v_1 is equal to one of the at most $q^2 + q$ values already in Π_1 is at most $\frac{q^2+q}{N-q}$. We now argue that $\Pi_1(+, u_1)$ is also undefined with high probability. For this, we show that the query $E(-, k, y)$ made by the simulator to wrap around when evaluating the IEM construction forward is fresh, i.e., it never made this query before nor received y as answer to a previous query $E(+, k, x)$. Assume that this does not hold. Then this means that such a query previously occurred when completing another chain (v'_2, u'_3) . But since we assumed that no value was overwritten in the tables before this call to $\text{CompleteChain}(v_2, u_3, 1)$, it can easily be seen that this implies that $(v'_2, u'_3) = (v_2, u_3)$, which cannot be since the simulator completes any chain at most once by construction. This implies that the value x returned by E comes at random from a set of size at least $N - q^2$ (since by Lemma 4 the simulator makes at most q^2 queries to E), so that $u_1 := x \oplus k$ is equal to one of the at most $q^2 + q$ values already in table Π_1 with probability at most $\frac{q^2+q}{N-q^2}$. Hence, summing over the at most q^2 calls to CompleteChain , we obtain

$$\begin{aligned} \Pr[\text{BadAdapt} | \neg \text{BadRand}] &\leq \sum_{i=1}^{q^2} \Pr[\text{BadAdapt}_i | \\ &\quad \neg \text{BadRand} \wedge \neg \text{BadAdapt}_j, j = 1, \dots, i-1] \\ &\leq q^2 \left(\frac{q^2+q}{N-q} + \frac{q^2+q}{N-q^2} \right) \leq \frac{8q^4}{N}. \end{aligned} \tag{11}$$

Combining (10) and (11) yields the result. \square

Lemma 6. *For any distinguisher \mathcal{D} of total oracle query cost at most q , one has*

$$\left| \Pr[\mathcal{D}^{\Sigma_1(E,P)} = 1] - \Pr[\mathcal{D}^{\Sigma_2(E,P)} = 1] \right| \leq \frac{16q^4}{N},$$

where both probabilities are taken over $E \leftarrow_{\$} \text{BC}(n, n), P \leftarrow_{\$} (\mathcal{P}_n)^4$.

Proof. Deferred to the full version of the paper [12] for reasons of space. \square

TRANSITION FROM Σ_2 TO Σ_3 AND RANDOMNESS MAPPING. We now consider the transition from the second to the third system, using a randomness mapping argument similar to the one of [19, 24]. For this, we define a map Λ mapping pairs (E, P) either to the special symbol \perp when (E, P) is bad, or to a tuple of *partial permutations* $P' = (P'_1, \dots, P'_4)$ when (E, P) is good. A partial permutation is a function $P'_i : \{+, -\} \times \{0, 1\}^n \rightarrow \{0, 1\}^n \cup \{*\}$ such that for all $u, v \in \{0, 1\}^n$, $P'_i(+, u) = v \neq * \Leftrightarrow P'_i(-, v) = u \neq *$.

The map Λ is defined for good pairs (E, P) as follows: run $\mathcal{D}^{\Sigma_2(E,P)}$, and consider the tables Π_i of the simulator at the end of the execution; then fill all undefined entries of the Π_i 's with the special symbol $*$. The result is exactly

$\Lambda(E, P)$. Since for a good pair (E, P) , the simulator never overwrites an entry in its tables, it follows that $\Lambda(E, P)$ is a tuple of partial permutations as just defined above. We say that a tuple of partial permutations $P' = (P'_1, \dots, P'_4)$ is good if it has a good preimage by Λ . We say that a tuple of permutations $P = (P_1, \dots, P_4)$ extends a tuple of partial permutations $P' = (P'_1, \dots, P'_4)$, denoted $P \vdash P'$, if for each $1 \leq i \leq 4$, P_i and P'_i agree on all entries such that $P'_i(\delta, w) \neq *$.

Lemma 7. *For any distinguisher \mathcal{D} of total oracle query cost at most q , one has*

$$\left| \Pr \left[\mathcal{D}^{\Sigma_2(E, P)} = 1 \right] - \Pr \left[\mathcal{D}^{\Sigma_3(P)} = 1 \right] \right| \leq \frac{52q^4}{N},$$

where the first probability is taken over $E \leftarrow_{\$} \text{BC}(n, n), P \leftarrow_{\$} (\mathcal{P}_n)^4$, and the second over $P \leftarrow_{\$} (\mathcal{P}_n)^4$.

Proof. Deferred to the full version of the paper [12] for reasons of space. \square

CONCLUDING. The proof of Theorem 5 directly follows by combining Lemmas 4, 6, and 7. As a corollary, we obtain from Theorem 4 that for any (q^2, ε) -evasive relation \mathcal{R} , the 4-round IEM cipher is $(q, \varepsilon + \mathcal{O}(q^4/2^n))$ -correlation intractable with respect to \mathcal{R} . Using again Example 1, the Davies-Meyer compression function based on the 4-round IEM cipher is $(q, \mathcal{O}(q^4/2^n))$ -preimage resistant in the Random Permutation Model. This is quite a weak security guarantee, and as already explained, this motivates the search for a block cipher construction (potentially the IEM cipher with a sufficient number of rounds) which is $(q, \sigma, t, \varepsilon)$ -seq-indifferentiable from an ideal cipher with $\sigma = \mathcal{O}(q)$ and $\varepsilon = \mathcal{O}(q/2^n)$.

A Formal Description of the Simulator

```

1  Simulator  $\mathcal{S}(P)$ :
2  Variables:
3    tables  $\Pi_1, \dots, \Pi_4$ , initially empty

4  public procedure Query( $i, \delta, w$ ):
5    if  $(\delta, w) \notin \Pi_i$  then
6       $w' := P_i(\delta, w)$ 
7       $\Pi_i(\delta, w) := w'$ 
8       $\Pi_i(\bar{\delta}, w') := w$             $\backslash\backslash$  may overwrite an entry
9       $\backslash\backslash$  complete newly created chains  $(v_2, u_3)$  if any
10     if  $i = 2$  then
11       if  $\delta = +$  then  $v_2 := w'$  else  $v_2 := w$ 
12       forall  $u_3 \in \Pi_3^+$  do
13         CompleteChain( $v_2, u_3, 1$ )
14     else if  $i = 3$  then
15       if  $\delta = +$  then  $u_3 := w$  else  $u_3 := w'$ 
16       forall  $v_2 \in \Pi_2^-$  do
17         CompleteChain( $v_2, u_3, 4$ )
18     return  $\Pi_i(\delta, w)$ 

19  private procedure CompleteChain( $v_2, u_3, \ell$ ):
20     $k := v_2 \oplus u_3$ 
21    case  $\ell = 1$ :                                34    case  $\ell = 4$ :
22       $\backslash\backslash$  evaluate the chain bw. up to  $v_1$  35       $\backslash\backslash$  evaluate the chain fw. up to  $u_4$ 
23       $u_2 := \Pi_2(-, v_2)$                       36       $v_3 := \Pi_3(+, u_3)$ 
24       $v_1 := u_2 \oplus k$                       37       $u_4 := v_3 \oplus k$ 
25       $\backslash\backslash$  evaluate the chain fw. up to  $u_1$  38       $\backslash\backslash$  evaluate the chain bw. up to  $v_4$ 
26       $v_3 := \Pi_3(+, u_3)$                       39       $u_2 := \Pi_2(-, v_2)$ 
27       $u_4 := v_3 \oplus k$                       40       $v_1 := u_2 \oplus k$ 
28       $v_4 := \text{Query}(4, +, u_4)$           41       $u_1 := \text{Query}(1, -, v_1)$ 
29       $y := v_4 \oplus k$                       42       $x := u_1 \oplus k$ 
30       $x := E(-, k, y)$                       43       $y := E(+, k, x)$ 
31       $u_1 := x \oplus k$                       44       $v_4 := y \oplus k$ 
32       $\backslash\backslash$  adapt the chain                  45       $\backslash\backslash$  adapt the chain
33      ForceVal( $u_1, v_1, 1$ )                 46      ForceVal( $u_4, v_4, 4$ )

47  private procedure ForceVal( $u_i, v_i, i$ ):
48     $\Pi_i(+, u_i) := v_i$             $\backslash\backslash$  may overwrite an entry
49     $\Pi_i(-, v_i) := u_i$             $\backslash\backslash$  may overwrite an entry

```

B Known-Key Attacks

Andreeva *et al.* [2], in an attempt to formalize known-key attacks, have introduced the notion of known-key indifferentiability (KK-indifferentiability), and shown that the 1-round Even-Mansour cipher is KK-indifferentiable from an ideal cipher. KK-indifferentiability for a block cipher construction \mathcal{C}^F is defined in a similar way as (full) indifferentiability, except that a random key k is drawn at the beginning of the security experiment, and the distinguisher is restricted to querying the construction \mathcal{C}^F in the real world or the ideal cipher E in the ideal world with the key k . Moreover, in the ideal world, the simulator is given the key k as input.

We argue however that the notion of [2] is slightly too restrictive to fully capture known-key attacks, because their definition involves only one single random key. If one tries to consider attacks with larger key arity, then the 1-round Even-Mansour cipher is *not* secure against known-key attacks. Consider the following simple example of a known-key attack against the 1-round Even-Mansour cipher (with identical round keys) involving two random keys. The adversary receives two random keys $k \neq k'$. It picks an arbitrary $x \in \{0, 1\}^n$ and defines $x' = x \oplus k \oplus k'$. Let $y = \text{EM}_k^P(x)$ and $y' = \text{EM}_{k'}^P(x')$. Then one can easily check that $x \oplus x' = y \oplus y'$. Yet for an ideal cipher E , given two random keys $k \neq k'$, finding two pairs (x, y) and (x', y') such that $E_k(x) = y$, $E_{k'}(x') = y'$, and $x \oplus x' = y \oplus y'$ can be shown to be hard: more precisely, an adversary making at most q queries to E finds such pairs with probability $\mathcal{O}(\frac{q^2}{2^n})$. In other words, for the 1-round EM construction, the adversary can very easily find a binary relation which is $(q, \mathcal{O}(\frac{q^2}{2^n}))$ -evasive with respect to an ideal cipher and involves the two “challenge” keys k, k' .

It is straightforward to extend the KK-indifferentiability definition given by [2] to handle larger key arity, by restricting the distinguisher to query its left oracle (\mathcal{C}^F/E) on a set of at most m keys k_1, \dots, k_m randomly drawn at the beginning of the experiment. Then, for $m > 1$, the 1-round IEM cipher is not KK-indifferentiable from an ideal cipher under this definition, as shown by the attack outlined above.

Similarly, one could easily modify the definition of correlation intractability (cf. Definition 3) in order to better capture the known-key setting, by simply drawing m' random keys $k_1, \dots, k_{m'}$ given as input to \mathcal{M}^F , and imposing to \mathcal{M} that its output $(\alpha_1, \dots, \alpha_m)$ only involves the “challenge” keys $k_1, \dots, k_{m'}$.

We leave the study of these new notions to future work.

References

1. Andreeva, E., Bogdanov, A., Dodis, Y., Mennink, B., Steinberger, J.P.: On the indifferentiability of key-alternating ciphers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 531–550. Springer, Heidelberg (2013). <http://eprint.iacr.org/2013/061>

2. Andreeva, E., Bogdanov, A., Mennink, B.: Towards Understanding the Known-Key Security of Block Ciphers. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 348–366. Springer, Heidelberg (2014)
3. Bellare, M., Cash, D.: Pseudorandom Functions and Permutations Provably Secure against Related-Key Attacks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 666–684. Springer, Heidelberg (2010)
4. Bellare, M., Kohno, T.: A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (2003)
5. Biham, E., Dunkelman, O., Keller, N.: Related-Key Boomerang and Rectangle Attacks. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 507–525. Springer, Heidelberg (2005)
6. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and Related-Key Attack on the Full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
7. Black, J.A., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002)
8. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.-X., Steinberger, J., Tischhauser, E.: Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 45–62. Springer, Heidelberg (2012)
9. Canetti, R., Goldreich, O., Halevi, S.: The Random Oracle Methodology, Revisited (Preliminary Version). In: Symposium on Theory of Computing, STOC 1998, pp. 209–218. ACM (1998). Full version available at <http://arxiv.org/abs/cs.CR/0010019>
10. Chen, S., Lampe, R., Lee, J., Seurin, Y., Steinberger, J.: Minimizing the Two-Round Even-Mansour Cipher. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 39–56. Springer, Heidelberg (2014). <http://eprint.iacr.org/2014/443>
11. Chen, S., Steinberger, J.: Tight Security Bounds for Key-Alternating Ciphers. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 327–350. Springer, Heidelberg (2014). <http://eprint.iacr.org/2013/222>
12. Cogliati, B., Seurin, Y.: On the Provable Security of the Iterated Even-Mansour Cipher against Related-Key and Chosen-Key Attacks. Full version of this paper. Available at <http://eprint.iacr.org/2015/069>
13. Daemen, J.: Limitations of the Even-Mansour construction. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 495–498. Springer, Heidelberg (1993)
14. Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging Merkle-Damgård for Practical Applications. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 371–388. Springer, Heidelberg (2009)
15. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 336–354. Springer, Heidelberg (2012)
16. Even, S., Mansour, Y.: A Construction of a Cipher from a Single Pseudorandom Permutation. Journal of Cryptology **10**(3), 151–162 (1997)
17. Farshim, P., Procter, G.: The Related-Key Security of Iterated Even-Mansour Ciphers. In: Fast Software Encryption, FSE 2015 (to appear, 2015). Full version available at <http://eprint.iacr.org/2014/953>

18. Goldenberg, D., Liskov, M.: On Related-Secret Pseudorandomness. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 255–272. Springer, Heidelberg (2010)
19. Holenstein, T., Künzler, R., Tessaro, S.: The Equivalence of the Random Oracle Model and the Ideal Cipher Model, Revisited. In: Symposium on Theory of Computing, STOC 2011, pp. 89–98. ACM (2011). Full version available at <http://arxiv.org/abs/1011.1264>
20. Iwata, T., Kohno, T.: New Security Proofs for the 3GPP Confidentiality and Integrity Algorithms. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 427–445. Springer, Heidelberg (2004)
21. Kiltz, E., Pietrzak, K., Szegedy, M.: Digital Signatures with Minimal Overhead from Indifferentiable Random Invertible Functions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 571–588. Springer, Heidelberg (2013)
22. Knudsen, L.R., Rijmen, V.: Known-Key Distinguishers for Some Block Ciphers. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 315–324. Springer, Heidelberg (2007)
23. Lampe, R., Patarin, J., Seurin, Y.: An Asymptotically Tight Security Analysis of the Iterated Even-Mansour Cipher. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 278–295. Springer, Heidelberg (2012)
24. Lampe, R., Seurin, Y.: How to Construct an Ideal Cipher from a Small Set of Public Permutations. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 444–463. Springer, Heidelberg (2013). <http://eprint.iacr.org/2013/255>
25. Lucks, S.: Ciphers Secure against Related-Key Attacks. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 359–370. Springer, Heidelberg (2004)
26. Mandal, A., Patarin, J., Seurin, Y.: On the Public Indifferentiability and Correlation Intractability of the 6-Round Feistel Construction. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 285–302. Springer, Heidelberg (2012). <http://eprint.iacr.org/2011/496>
27. Maurer, U.M., Renner, R.S., Holenstein, C.: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 21–39. Springer, Heidelberg (2004)
28. Nyberg, K., Knudsen, L.R.: Provable Security against Differential Cryptanalysis. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 566–574. Springer, Heidelberg (1993)
29. Patarin, J.: The “Coefficients H” Technique. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 328–345. Springer, Heidelberg (2009)
30. Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with Composition: Limitations of the Indifferentiability Framework. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 487–506. Springer, Heidelberg (2011)
31. Rogaway, P.: Formalizing Human Ignorance. In: Nguyêñ, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 211–228. Springer, Heidelberg (2006)
32. Steinberger, J.: Improved Security Bounds for Key-Alternating Ciphers via Hellinger Distance. IACR Cryptology ePrint Archive, Report 2012/481 (2012). Available at <http://eprint.iacr.org/2012/481>
33. Winternitz, R.S.: A Secure One-Way Hash Function Built from DES. In: IEEE Symposium on Security and Privacy, pp. 88–90 (1984)
34. Yoneyama, K., Miyagawa, S., Ohta, K.: Leaky Random Oracle. IEICE Transactions 92-A(8), 1795–1807 (2009)

Fully Homomorphic Encryption II

FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second

Léo Ducas^{1(✉)} and Daniele Micciancio²

¹ Centrum Wiskunde and Informatica, Amsterdam, Netherlands
leo.ducas@cwi.nl

² University of California, San Diego, California, USA
daniele@cse.ucsd.edu

Abstract. The main bottleneck affecting the efficiency of all known fully homomorphic encryption (FHE) schemes is Gentry’s bootstrapping procedure, which is required to refresh noisy ciphertexts and keep computing on encrypted data. Bootstrapping in the latest implementation of FHE, the HElib library of Halevi and Shoup (Crypto 2014), requires about six minutes. We present a new method to homomorphically compute simple bit operations, and refresh (bootstrap) the resulting output, which runs on a personal computer in just about half a second. We present a detailed technical analysis of the scheme (based on the worst-case hardness of standard lattice problems) and report on the performance of our prototype implementation.

1 Introduction

Since Gentry’s discovery of the first fully homomorphic encryption (FHE) scheme [15], much progress has been made both towards basing the security of FHE on more standard and well understood security assumptions, and improving the efficiency of Gentry’s initial solution.

On the security front, a sequence of papers [2, 5, 8, 9, 16] has lead to (leveled) FHE schemes based on essentially the same intractability assumptions underlying standard (non homomorphic) lattice based encryption. To date, the main open theoretical problem still left to be solved is how to remove the “circular security” assumption made in [15] (and all subsequent works) to turn a leveled FHE scheme (i.e., a scheme where the homomorphic computation depth is chosen at key generation time) into a full fledged one which allows to perform arbitrarily large homomorphic computations on encrypted data, even after all key material has been fixed.

This research was supported in part by the DARPA PROCEED program and NSF grant CNS-1117936. Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or NSF.

L. Ducas—This research was done while the author was employed by University of California, San Diego.

Improving the efficiency of Gentry’s scheme has received even more attention [1, 2, 6, 17–21, 23], resulting in enhanced asymptotic performance, and some reference implementations and libraries [17, 23] that are efficient enough to be run on a personal computer. Still, the cost of running FHE schemes is quite substantial. The main bottleneck is caused by the fact that all current FHE solutions are based on “noisy” encryption schemes (based on lattices or similar problems) where homomorphic operations increase the noise amount and lower the quality of ciphertexts. As more homomorphic operations are performed, the noise can easily grow to a level where the ciphertexts are no longer decryptable, and operating on them produces meaningless results. Gentry’s breakthrough discovery [15] was an ingenious “bootstrapping” technique (used in all subsequent works) that refreshes the ciphertexts by homomorphically computing the decryption function on encrypted secret key, and bringing the noise of the ciphertexts back to acceptable levels. This bootstrapping method allows to homomorphically evaluate arbitrary circuits, but it is also the main bottleneck in any practical implementation due to the complexity of homomorphic decryption.

Going back to efficiency considerations, the current state of the art in terms of FHE implementation is represented by the recent HElib of Halevi and Shoup [23, 24], which reported a bootstrapping/refreshing procedure with running times around 6 minutes. While this is much better than previous implementations, and a nontrivial amount of computation can be performed in-between refreshing operations, the fact that even the simplest computation requiring bootstrapping takes such a macroscopic amount of time makes FHE clearly unattractive.

Our Work. The goal of this paper is to investigate to what extent the running time of a useful FHE bootstrapping procedure can be reduced. We do so by analyzing bootstrapping *in vitro*, i.e., in the simplest possible setting: given two encrypted bits $E(b_1)$ and $E(b_2)$, we want to compute their logical NAND (or any other complete boolean operation) and obtain the encrypted result $E(b_1 \bar{\wedge} b_2)$ in a form which is similar to the input bits. As in the most recent FHE schemes, here $E(\cdot)$ is just a standard lattice (LWE [32]) encryption scheme. In particular, $E(b_i)$ are noisy encryptions, and the output ciphertext $E(b_1 \bar{\wedge} b_2)$ is homomorphically decrypted (i.e., bootstrapped) in order to reduce its noise level back to that of $E(b_1)$ and $E(b_2)$. Our main result is a new bootstrapping method and associated implementation that allows to perform the entire computation (consisting of homomorphic NAND computation and homomorphic decryption/bootstrapping) in less than a second on a standard (consumer grade) personal computer as detailed in Section 6.4.

We remark that the problem solved here is definitely simpler than HElib [23], as we perform only a single bit operation before bootstrapping, while [23] allows to perform more complex operations. In fact, using complex ciphertexts packing and homomorphic SIMD techniques, [23] achieves an amortized cost (per homomorphic bit operation) which we estimate to be in the same order of magnitude as our solution. The main improvement with respect to previous work is in terms of granularity and simplicity: we effectively show that half hour delays are not a necessary requirement of bootstrapped FHE computations,

and bootstrapping itself can be achieved at much higher speeds than previously thought possible. Another attractive feature of the scheme presented in this paper is simplicity: we implemented our fully bootstrapped NAND computation in just a few hundreds lines of code and just a few days of programming effort.

Finally, our methods are not necessarily limited to a single NAND computation. As a simple extension of our basic scheme we show how to compute (homomorphically, and at essentially the same level of efficiency) various other operations, like majority, threshold gates. This extension also offers *xor-for-almost-free* as previous homomorphic schemes. Combining our fast (subsecond) bootstrapping method with other techniques that allow to perform substantially more complex computations in-between bootstrappings, is left as an open problem.

Techniques. Our improvement is based on two main techniques. One is a new method to homomorphically compute the NAND of two LWE encryptions. We recall that LWE encryption satisfies certain approximate additive homomorphic properties. Specifically, given two encryptions $E(m_1)$ and $E(m_2)$ one can compute a noisier version of $E(m_1 + m_2)$. When working modulo 2, this allows to homomorphically compute the exclusive-or of two bits. The way we extend this operation to a logical NAND computation is by moving (during bootstrapping) from arithmetic modulo 2 to arithmetic modulo 4. So, adding $E(m_1)$ and $E(m_2)$ results in the encryption $E(m)$ of $m = 2$ (if $m_1 \wedge m_2 = 0$) or $m \in \{0, 1\}$ (if $m_1 \wedge m_2 = 1$). Moving from this ciphertext to the encryption of $m_1 \wedge m_2$ is then achieved by a simple affine transformation.

The main advantage of our new homomorphic NAND operation is that it introduces a much lower level of noise than previous techniques. So, the refreshing procedure (required for bootstrapping) is faced with a much simpler task. Our second technical contribution builds on a recent method from [2] to implement and speed up bootstrapping. Decryption of LWE ciphertexts requires essentially the computation of a scalar product (modulo q) and a rounding operation. So, homomorphic decryption needs to compute these operations on encrypted data. The scheme of [2] uses a homomorphic cryptosystem that encrypts integers modulo q , and allows the efficient computation of scalar products. This is achieved using a homomorphic encryption scheme for binary messages $x \in \{0, 1\}$, and encoding elements $v \in C$ of a cyclic group as vectors of ciphertexts $E(x_1), \dots, E(x_{|C|})$, where $x_i = 1$ if and only if $i = v$. We introduce a ring variant of the bootstrapping method of [2] that also supports efficient homomorphic computation of scalar products modulo q . The use of ring lattices was first suggested¹ in [31] to reduce the asymptotic computation time of lattice cryptography from quadratic to quasi-linear (using FFT techniques), and have become a fundamental technique to bring theoretical lattice constructions to levels of performance that are attractive in practice. Our work uses the LWE instantiation of ring lattices [29, 30] for the efficient implementation of encryption. But our bootstrapping method goes beyond the use of ring

¹ Similar lattices had previously been used in practice also by the NTRU cryptosystem [25], but without employing quasi-linear FFT techniques, and no connection to the worst-case complexity of lattice problems.

lattices to speed up normal lattice operations. We also use the ring structure of these lattices to directly implement the encryption of cyclic groups by encoding the cyclic group \mathbb{Z}_q into the group of roots of unity: $i \mapsto X^i$ where i is a primitive q -th root of unity. This allows to implement a bootstrapping method similar to [2], but where each cyclic group element is encoded by a single ciphertext, rather than a vector of ciphertexts.

As a last technique, in order to contain noise generation during key switching operations, we use LWE instances with binary secrets, which were recently proved as hard as standard LWE in [7].

Like all previously known schemes, our FHE construction requires (in addition to standard worst-case lattice intractability assumptions) a circular security assumption in order to release a compact homomorphic evaluation key, and allow to combine an arbitrarily large number of homomorphic bit operations.

Organization. The rest of the paper is organized as follows. In section 2 we give some background on lattices and related techniques as used in the paper. In Section 3 we present a detailed description of the LWE encryption scheme that we want to bootstrap. The high level structure of our bootstrapped homomorphic NAND computation is given in Section 4. Section 5 goes into the core of our new refreshing procedure based on ring lattices. Section 6 describes concrete parameters, implementation and performance details. Section 7 concludes the paper with extensions and open problems.

2 Preliminaries

We will use bold-face lower-case letters $\mathbf{a}, \mathbf{b} \dots$ to denote column vectors over \mathbb{Z} or any other ring \mathcal{R} , and boldface upper-case letters $\mathbf{A}, \mathbf{B} \dots$ for matrices. The product symbol \cdot will be used for both scalar products of two column vectors, and for matrix product, to be interpreted as the only applicable one. The norm $\|\cdot\|$, will denote the euclidean norm. When speaking of the norm of a vector \mathbf{v} over the residue ring \mathbb{Z}_Q of \mathbb{Z} modulo Q , we mean the shortest norm among the equivalence class of $\mathbf{v} \in \mathbb{Z}_Q^n$ in \mathbb{Z}^n .

2.1 Distributions

A *randomized rounding* function $\chi: \mathbb{R} \rightarrow \mathbb{Z}$ is a function mapping each $x \in \mathbb{R}$ to a distribution over \mathbb{Z} such that $\chi(x+n) = \chi(x) + n$ for all integers n . For any $x \in \mathbb{R}$, the random variable $\chi(x) - x$ is called the rounding error of $\chi(x)$. As a special case, when the domain of χ is restricted to \mathbb{Z} , we have $\chi(x) = x + \chi(0)$, i.e., the randomized rounding function simply adds a fixed “noise” distribution $\chi(0)$ to the input $x \in \mathbb{Z}$.

A random variable X over \mathbb{R} is *subgaussian* with parameter $\alpha > 0$ if for all $t \in \mathbb{R}$, the (scaled) moment-generating function satisfies $E[\exp(2\pi tX)] \leq \exp(\pi\alpha^2 t^2)$. If X is subgaussian, then its tails are dominated by a Gaussian of parameter α , i.e., $\Pr\{|X| \geq t\} \leq 2 \exp(-\pi t^2/\alpha^2)$ for all $t \geq 0$. Any B -bounded

symmetric random variable X (i.e., $|X| \leq B$ always) is subgaussian with parameter $B\sqrt{2\pi}$. More generally, we say that a random vector \mathbf{x} (respectively, a random matrix \mathbf{X}) is subgaussian (of parameter α) if all its one-dimensional marginals $\langle \mathbf{u}, \mathbf{x} \rangle$ (respectively, $\mathbf{u}^t \mathbf{X} \mathbf{v}$) for unit vectors \mathbf{u}, \mathbf{v} are subgaussian (of parameter α). It follows immediately from the definition that the concatenation of independent subgaussian vectors with common parameter α , interpreted as either a vector or matrix, is subgaussian with parameter α .

2.2 The Cyclotomic Ring

Throughout the paper, we let N be a power of 2 defining the $(2N)$ th cyclotomic polynomial $\Phi_{2N}(X) = X^{2N} + 1$ and associated cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/(X^{2N} + 1)$. We also write $\mathcal{R}_Q = \mathcal{R}/(QR)$ for the residue ring of \mathcal{R} modulo an integer Q . Elements in \mathcal{R} have a natural representation as polynomials of degree $N - 1$ with coefficients in \mathbb{Z} , and \mathcal{R} can be identified (as an additive group) with the integer lattice \mathbb{Z}^N , where each ring element $a = a_0 + a_1x + \dots + a_{N-1}x^{N-1} \in \mathcal{R}$ is associated with the coefficient vector $\vec{a} = (a_0, \dots, a_{N-1}) \in \mathbb{Z}^N$. We extend the notation $\vec{\cdot}$ to any vector (or matrix) over \mathcal{R} component-wise. We use the identification $\mathcal{R} = \mathbb{Z}^N$ to define standard lattice quantities like the euclidean length of a ring element $\|a\| = \|\vec{a}\| = \sqrt{\sum_i |a_i|^2}$, or the spectral norm of a matrix $\mathbf{R} \in \mathcal{R}^{w \times k}$ of ring elements $s_1(\mathbf{R}) = \sup_{\mathbf{x} \in \mathcal{R}^k \setminus \{0\}} \|\mathbf{R} \cdot \mathbf{x}\|/\|\mathbf{x}\|$.

The ring \mathcal{R} is also identified with the sub-ring of anti-circulant square matrices of dimension N by regarding each ring element $r \in \mathcal{R}$ as a linear transformation $x \mapsto r \cdot x$ over (the coefficient embedding) of \mathcal{R} . The corresponding matrix is denoted $\vec{r} \in \mathbb{Z}^{N \times N}$, and its first column is \vec{r} . (The other columns are the cyclic rotations of \vec{r} with the cycled entries negated.) We extend the notation $\vec{\cdot}$ to vectors and matrices over \mathcal{R} : for $\mathbf{R} \in \mathcal{R}^{w \times k}$, $\vec{\mathbf{R}} \in \mathbb{Z}^{Nw \times Nk}$ is a matrix with anti-circulant $N \times N$ blocks. Notice that the definition of spectral norm of a ring element (or a matrix of ring elements) is consistent with the definition of spectral norm of the corresponding anticirculant matrix (or blockwise anti-circulant matrix): $s_1(r) = s_1(\vec{r})$ and $s_1(\mathbf{R}) = s_1(\vec{\mathbf{R}})$.

We say that a random polynomial a is subgaussian if its associated vector \vec{a} is subgaussian. The fact that a is subgaussian *does not* imply that its associated anticirculant matrix \vec{a} is also subgaussian, because its columns are not independent. Nevertheless, subgaussianity of a ring elements still allows a good bound on its singular norm. This bound is as small as its non-ring counterpart as soon as either w or k is larger than $\omega(\sqrt{\log N})$.

Fact 1 (Adapted from [12], Fact 6). *If \mathcal{D} is a subgaussian distribution of parameter α over \mathcal{R} , and $\mathbf{R} \leftarrow \mathcal{D}^{w \times k}$ has independent coefficients drawn from \mathcal{D} , then, with overwhelming probability, we have $s_1(\mathbf{R}) \leq \alpha\sqrt{N} \cdot O(\sqrt{w} + \sqrt{k} + \omega(\sqrt{\log N}))$.*

Invertibility in \mathcal{R} . Invertibility in cyclotomic rings has to be handled with care. (E.g., see [12].) The main issue is that, for a power-of-two cyclotomic ring $\mathcal{R} =$

$\mathbb{Z}[X]/(X^N + 1)$, the residue ring \mathcal{R}_Q is never a field whatever the choice of Q . Yet, for appropriate moduli Q , it is not so far from being a field. More concretely, for Q a power of 3 most elements in \mathcal{R} will be invertible, and so will most of the square matrices over \mathcal{R} as detailed by the following Lemma 4. The lemma uses the following two facts.

Fact 2 (Irreducible factors of $X^N + 1$ modulo 3). *For any $k \geq 3$ and $N = 2^k$ we have $X^N + 1 = (X^{N/2} + X^{N/4} - 1) \cdot (X^{N/2} - X^{N/4} - 1) \pmod{3}$ and both factors are irreducible in $\mathbb{F}_3[X]$.*

Proof. This follows directly from [26, Theorem 2.47].

Lemma 3 (Hensel Lemma for powers of prime integers). *Let \mathcal{R} be the ring $\mathbb{Z}[X]/(F(X))$ for some monic polynomial $F \in \mathbb{Z}[X]$. For any prime p , if $u \in \mathcal{R}_{p^e}$ is invertible \pmod{p} (i.e. it is invertible in \mathcal{R}_p) then u is also invertible in \mathcal{R}_{p^e} .*

Lemma 4 (Invertibility of random matrices). *For Q a power of 3, and any dimension k , if \mathcal{D} is a distribution over \mathcal{R}_Q such that $\mathcal{D} \pmod{3}$ is (statistically close to) uniform over \mathcal{R}_3 , then, with overwhelming probability $\mathbf{D} \leftarrow \mathcal{D}^{k \times k}$ is invertible.*

Proof. By Fact 2, the ring \mathcal{R}_3 factors as $\mathcal{R}_3 = \mathbb{F}_1 \times \mathbb{F}_2$, where $\mathbb{F}_1 = \mathcal{R}/(3, P_1(X) = X^{N/2} + X^{N/4} - 1)$ and $\mathbb{F}_2 = \mathcal{R}/(3, P_2(X) = X^{N/2} - X^{N/4} - 1)$ are fields of order $q = \#\mathbb{F}_i = 3^{N/2}$. Note that $\mathbf{D} \pmod{(3, P_i(X))}$ is (statistically close to) a uniform random variable over $\mathbb{F}_i^{k \times k}$. We recall that the number of invertible matrices over the field of size q is given by

$$\begin{aligned} \#\mathcal{GL}(k, q) &= q^{k^2} \prod_{i=0}^{k-1} (1 - q^{i-k}) \\ &\geq q^{k^2} \left(1 - \sum_{i=1}^k q^{-i}\right) \geq q^{k^2} \left(1 - \frac{1}{q} \sum_{i \geq 0} q^{-i}\right) = q^{k^2} \left(1 - \frac{1}{q-1}\right). \end{aligned}$$

In particular $\mathbf{D} \pmod{(3, P_i(X))}$ is invertible except with probability $1/(q-1)$. By a union bound, \mathbf{D} is invertible modulo both $(3, P_1(X))$ and $(3, P_2(X))$, except with negligible probability $2/(q-1) = 2/(3^{N/2}-1)$. It follows that \mathbf{D} is invertible modulo 3, and by Hensel lifting (Lemma 3), also modulo Q . Indeed, Hensel lemma extends to *matrices* over \mathcal{R} , considering that a matrix $\mathbf{M} \in \mathcal{R}_Q^{k \times k}$ is invertible if and only if its determinant over \mathcal{R}_Q is invertible.

3 LWE Symmetric Encryption

We recall the definition of the most basic LWE symmetric encryption scheme (see [3, 4, 32]). LWE symmetric encryption is parametrized by a dimension n , a message-modulus $t \geq 2$, a ciphertext modulus $q = n^{O(1)}$ and a randomized

rounding function $\chi: \mathbb{R} \rightarrow \mathbb{Z}$. The message space of the scheme is \mathbb{Z}_t . (Typically, the rounding function has error distribution $|\chi(x) - x| < q/2t$, and $t = 2$ is used to encrypt message bits.) The (secret) key of the encryption scheme is a vector $\mathbf{s} \in \mathbb{Z}_q^n$, which may be chosen uniformly at random, or as a random short vector. The encryption of a message $m \in \mathbb{Z}_t$ under key $\mathbf{s} \in \mathbb{Z}_q^n$ is

$$\text{LWE}_{\mathbf{s}}^{t/q}(m) = (\mathbf{a}, \chi(\mathbf{a} \cdot \mathbf{s} + mq/t) \bmod q) \in \mathbb{Z}_q^{n+1} \quad (1)$$

where $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ is chosen uniformly at random. Notice that when t divides q , the encryption of m equals $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e + mq/t \bmod q)$, where the error e is chosen according to a fixed noise distribution $\chi(0)$. A ciphertext (\mathbf{a}, b) is decrypted by computing

$$m' = \lfloor t(b - \mathbf{a} \cdot \mathbf{s})/q \rfloor \bmod t \in \mathbb{Z}_t. \quad (2)$$

We write $\text{LWE}_{\mathbf{s}}^{t/q}(m)$ to denote the set of all possible encryptions of m under \mathbf{s} . The error of a ciphertext $(\mathbf{a}, b) \in \text{LWE}_{\mathbf{s}}^{t/q}(m)$ is the random variable $\text{err}(\mathbf{a}, b) = (b - \mathbf{a} \cdot \mathbf{s} - mq/t) \bmod q$ describing the rounding error, reduced modulo q to the centered interval $[-q/2, q/2]$. Notice that the error $\text{err}(\mathbf{a}, b)$ depends not just on (\mathbf{a}, b) , but also on \mathbf{s}, q, t and m . Also, in the absence of any restriction on the error, a ciphertext $(\mathbf{a}, b) \in \text{LWE}_{\mathbf{s}}^{t/q}(m)$ can be any vector in \mathbb{Z}_q^{n+1} . We write $\text{LWE}_{\mathbf{s}}^{t/q}(m, E)$ to denote the set of all ciphertexts $\mathbf{c} \in \text{LWE}_{\mathbf{s}}^{t/q}(m)$ with error bounded by $|\text{err}(\mathbf{c})| < E$. It is easy to check that for all $(\mathbf{a}, b) \in \text{LWE}_{\mathbf{s}}^{t/q}(m, q/2t)$, the decryption procedure correctly recovers the encrypted message:

$$\lfloor t(b - \mathbf{a} \cdot \mathbf{s}) \rfloor \bmod t = \left\lfloor \frac{t}{q} \cdot \left(\frac{q}{t} m + e \right) \right\rfloor = \left\lfloor m + \frac{t}{q} e \right\rfloor = m \bmod t$$

because $\frac{t}{q}|e| < 1/2$.

Modulus Switching. LWE ciphertexts can be converted from one modulus Q to another q using the (scaled) randomized rounding function $[\cdot]_{Q:q}: \mathbb{Z}_Q \rightarrow \mathbb{Z}_q$ defined as

$$[x]_{Q:q} = \lfloor qx/Q \rfloor + B$$

where $B \in \{0, 1\}$ is a Bernoulli random variable with $\Pr\{B = 1\} = (qx/Q) - \lfloor qx/Q \rfloor \in [0, 1]$. Notice that $\mathbb{E}[x]_{Q:q} = \lfloor qx/Q \rfloor + \mathbb{E}[B] = qx/Q$ and $|[x]_{Q:q} - (qx/Q)| < 1$ with probability 1. In particular, the rounding error $[x]_{Q:q} - (qx/Q)$ is subgaussian of parameter $\sqrt{2\pi}$. The randomized rounding function is applied to vectors (e.g., LWE ciphertexts) coordinatewise:

$$\text{ModSwitch}(\mathbf{a}, b) = [(\mathbf{a}, b)]_{Q:q} = (([a_1]_{Q:q}, \dots, [a_n]_{Q:q}), [b]_{Q:q}). \quad (3)$$

Lemma 5. *For any $\mathbf{s} \in \mathbb{Z}_q^n$, $m \in \mathbb{Z}_t$ and ciphertext $\mathbf{c} \in \text{LWE}_{\mathbf{s}}^{t/Q}(m)$ with subgaussian error of parameter σ , the rounding $\text{ModSwitch}(\mathbf{c}) = [\mathbf{c}]_{Q:q}$ is a $\text{LWE}_{\mathbf{s}}^{t/q}(m)$ ciphertext with subgaussian error of parameter $\sqrt{(q\sigma/Q)^2 + 2\pi(\|\mathbf{s}\|^2 + 1)}$.*

Proof. Let $\mathbf{c} = (\mathbf{a}, b)$ and $[\mathbf{c}]_{Q:q} = (\mathbf{a}', b')$. We have $a'_i = \frac{q}{Q}a_i + r_i$ and $b' = \frac{q}{Q}b + r_0$ for independent subgaussian rounding errors $r_0 \dots r_n$ of parameter $\sqrt{2\pi}$. It follows that \mathbf{c}' is an $\text{LWE}_{\mathbf{s}}^{t/q}$ encryption of m with error $\text{err}(\mathbf{c}') = b' - \mathbf{a}' \cdot \mathbf{s} - \frac{qm}{t} = (q\text{err}(\mathbf{c})/Q) + r_0 - \sum_{i=1}^n s_i r_i$. Since $\text{err}(\mathbf{c}), r_0, \dots, r_n$ are independent subgaussian variables, their sum is also subgaussian, with parameter $\sqrt{(q\sigma/Q)^2 + 2\pi(||\mathbf{s}||^2 + 1)}$.

In practice, one may use the non-randomized rounding function $\lfloor \cdot \rfloor$. Then, the error of the output of `ModSwitch`, according to the central limit heuristic is expected to be close to a gaussian of standard deviation $\sqrt{(q\sigma/Q)^2 + (||\mathbf{s}||^2 + 1)/12}$, based on the randomness of \mathbf{a} . The factor $1/12$ comes from the standard deviation of a uniform distribution in $[-\frac{1}{2}, \frac{1}{2}]$.

Key Switching. Key switching allows to convert an LWE encryption under a key $\mathbf{z} \in \mathbb{Z}_q^N$ into an LWE encryption of the same message (and slightly larger error) under a different key $\mathbf{s} \in \mathbb{Z}_q^n$. The key switching procedure is parametrized by a base B_{ks} , and requires as an auxiliary input a suitable encryption of \mathbf{z} under \mathbf{s} . Specifically, let $\mathbf{k}_{i,j,v} \in \text{LWE}_{\mathbf{s}}^{q/q}(vz_i B_{\text{ks}}^j)$ be an encryption of $vz_i B_{\text{ks}}^j$ under \mathbf{z} , for all $i = 1, \dots, N$, $v \in \{0, \dots, B_{\text{ks}}\}$ and $j = 0, \dots, d_{\text{ks}} - 1$, where $d_{\text{ks}} = \lceil \log_{B_{\text{ks}}} q \rceil$. (Notice that the message $vz_i B_{\text{ks}}^j$ is interpreted as a value modulo $t = q$, and therefore the ciphertext $\mathbf{k}_{i,j,v}$ is not typically decryptable because it has error bigger than $q/2t = 1/2$.) Given the switching key $\mathfrak{K} = \{\mathbf{k}_{i,j,v}\}$ and a ciphertext $(\mathbf{a}, b) \in \text{LWE}_{\mathbf{z}}^{t/q}(m)$, the key switching procedure computes the base- B_{ks} expansion of each coefficient $a_i = \sum_j a_{i,j} B_{\text{ks}}^j$, and outputs

$$\text{KeySwitch}((\mathbf{a}, b), \mathfrak{K}) = (\mathbf{0}, b) - \sum_{i,j} \mathbf{k}_{i,j,a_{i,j}}. \quad (4)$$

Lemma 6. *The key switching procedure, given a ciphertext $\mathbf{c} \in \text{LWE}_{\mathbf{z}}^{t/q}(m)$ with subgaussian error of parameter α , and switching keys $\mathbf{k}_{i,j,v} = \text{LWE}_{\mathbf{s}}^{q/q}(vz_i B_{\text{ks}}^j)$ and subgaussian error of parameter σ , outputs an encryption $\text{KeySwitch}(\mathbf{c}, \{\mathbf{k}_{i,j,v}\}) \in \text{LWE}_{\mathbf{z}}^{t/q}(m)$ with subgaussian error of parameter $\sqrt{\alpha^2 + Nd_{\text{ks}}\sigma^2}$.*

Proof. Let $e_{i,j,v} = \text{err}(\mathbf{k}_{i,j,v})$, so that $\mathbf{k}_{i,j,v} = (\mathbf{a}'_{i,j,v}, \mathbf{a}'_{i,j,v} \cdot \mathbf{s} + vz_i B_{\text{ks}}^j + e_{i,j,v})$ for some $\mathbf{a}'_{i,j,v} \in \mathbb{Z}_q^n$. The output of the key switching procedure is $\text{KeySwitch}(\mathbf{a}, b) = (\mathbf{a}', b')$ where $\mathbf{a}' = -\sum_{i,j} \mathbf{a}'_{i,j,a_{i,j}}$ and

$$b' = b - \sum_{i,j} (\mathbf{a}'_{i,j,a_{i,j}} \cdot \mathbf{s} + a_{i,j} z_i B_{\text{ks}}^j + e_{i,j,a_{i,j}}) = b - \mathbf{a} \cdot \mathbf{z} + \mathbf{a}' \cdot \mathbf{s} - E,$$

where $E = \sum_{i,j} e_{i,j,a_{i,j}}$ is subgaussian with parameter $\sigma\sqrt{Nd_{\text{ks}}}$. It follows that (\mathbf{a}', b') has error

$$\text{err}(\mathbf{a}', b') = b' - \mathbf{a}' \cdot \mathbf{s} - \frac{qm}{t} = b - \mathbf{a} \cdot \mathbf{z} - E - \frac{qm}{t} = \text{err}(\mathbf{a}, b) - E.$$

Since $\text{err}(\mathbf{a}, b)$ and E are both subgaussian, their difference is also subgaussian with parameter $\sqrt{\alpha^2 + Nd_{\text{ks}}\sigma^2}$.

4 Our FHE: High Level Structure

In this section we describe the high level structure/design of our fully homomorphic (symmetric) encryption scheme. (This private-key FHE scheme can be transformed into a public-key one using standard techniques.) The encryption scheme itself is just the standard LWE symmetric encryption described in Section 3. For now we focus on encrypting single bits, and evaluating boolean NAND circuits. In summary, we need to solve the following problem: given two ciphertexts $c_i \in \text{LWE}_s^{2/q}(m_i)$ (for $i = 0, 1$), compute a ciphertext $c \in \text{LWE}_s^{2/q}(m)$ where $m = 1 - m_0 \cdot m_1 = m_0 \bar{\wedge} m_1$ is the logical NAND of m_0 and m_1 .

4.1 A New Homomorphic NAND Gate

The main idea to perform this encrypted NAND computation is to assume that the input ciphertexts are available in a slightly different form. (We will see later how to perform the required transformation.) Namely, assume that the input bits $m_0, m_1 \in \{0, 1\}$ are encrypted as ciphertexts $c_i \in \text{LWE}_s^{4/q}(m_i, q/16)$ using a slightly different message modulus $t = 4$ and error bound $E = q/16$. (Compare to the standard binary LWE encryption parameters $t = 2$ and $E = q/4$.)

Lemma 7. *There is a simple algorithm*

$$\text{HomNAND}: \text{LWE}_s^{4/q}(m_0, q/16) \times \text{LWE}_s^{4/q}(m_1, q/16) \rightarrow \text{LWE}_s^{2/q}(m_0 \bar{\wedge} m_1, q/4)$$

that on input two ciphertexts $\mathbf{c}_i \in \text{LWE}_s^{4/q}(m_i, q/16)$ (for $i = 0, 1$) encrypting binary messages $m_0, m_1 \in \{0, 1\}$, outputs an encryption $\text{HomNAND}(\mathbf{c}_0, \mathbf{c}_1) \in \text{LWE}_s^{2/q}(m, q/4)$ of their logical NAND $m = 1 - m_0 m_1 = m_0 \bar{\wedge} m_1$ with error less than $q/4$.

Proof. The NAND of the two ciphertexts $c_i = (\mathbf{a}_i, b_i)$ can be easily computed as

$$(\mathbf{a}, b) = \text{HomNAND}((\mathbf{a}_0, b_0), (\mathbf{a}_1, b_1)) = \left(-\mathbf{a}_0 - \mathbf{a}_1, \frac{5q}{8} - b_0 - b_1 \right).$$

(Remember that we assumed for simplicity that $8 = 2t$ divides q , and therefore $5q/8$ is an integer.) The resulting ciphertext satisfies

$$b - \mathbf{a}s - (1 - m_0 m_1) \frac{q}{2} = \frac{q}{4} \left(\frac{1}{2} - (m_0 - m_1)^2 \right) - (e_0 + e_1) = \pm \frac{q}{8} - (e_0 + e_1).$$

So, $(\mathbf{a}, b) = \text{HomNAND}(c_0, c_1)$ is a regular $\text{LWE}_s^{2/q}$ encryption of $1 - m_0 m_1 = m_0 \bar{\wedge} m_1$ with error at most

$$\left| \pm \frac{q}{8} - (e_0 + e_1) \right| < \frac{q}{8} + \frac{q}{16} + \frac{q}{16} = \frac{q}{4}.$$

Notice that the **HomNAND** function can be computed from the input ciphertexts without using any key material, and it requires just a handful of additions modulo q . This shows that in order to compute the NAND of two ciphertexts (and therefore homomorphically evaluate any boolean circuit on LWE encryptions), it is enough to be able to compute a refreshing function

$$\text{Refresh: } \text{LWE}_{\mathbf{s}}^2(m, q/4) \rightarrow \text{LWE}_{\mathbf{s}}^4(m, q/16).$$

The refreshing function will require some key material, and it will be substantially more expensive of **HomNAND**, accounting essentially for the whole cost of homomorphic circuit evaluation. Notice that the number of refresh computations required to evaluate a circuit with g gates is $n + g$ (one for each circuit input and gate-output wire). Assuming that the encrypted input bits are already provided in refreshed form (e.g., by using the modified $\text{LWE}^{4/q}(m, q/16)$ encryption scheme), one needs just 1 refresh evaluation per gate, applied to the output of the gate, rather than 2 evaluation (one for each input into the gate). So, the computational cost of homomorphically evaluating a NAND gate is essentially that of a single refresh function computation.

Improvement. Previous methods to compute homomorphic AND gates on LWE ciphertexts require errors of input to be at most $O(\sqrt{q})$, against $O(q)$ in our case. Our technique therefore relaxes the requirement on the Refresh procedure, potentially making the overall scheme faster.

4.2 Refreshing via Homomorphic Accumulator

We now move to the description of the refreshing function. As in all previous works on FHE, our ciphertext refreshing is based on Gentry’s bootstrapping technique of homomorphically evaluating the decryption function. More specifically, in our setting, given an LWE ciphertext $(\mathbf{a}, b) \in \text{LWE}_{\mathbf{s}}^{2/q}(m)$, we compute an encryption $E(m)$ of the same message under a different encryption scheme E by homomorphically evaluating the LWE decryption procedure (2) on the encrypted key $E(\mathbf{s})$ to yield

$$\lfloor 2(b - \mathbf{a} \cdot E(\mathbf{s}))/q \rfloor \bmod 2 \simeq E(m).$$

We recall that the final goal of the refreshing function is to obtain an encryption in $\text{LWE}^{4/q}(m, q/16)$. However, this target encryption scheme is not versatile enough to perform the required homomorphic computation. Instead, following [2], we use an intermediate encryption scheme E with message space \mathbb{Z}_q , which allows to encrypt the secret $\mathbf{s} \in \mathbb{Z}_q^n$ componentwise $E(\mathbf{s}) = (E(s_1), \dots, E(s_n))$ and supports the efficient computation of affine transformations $b - \mathbf{a} \cdot E(\mathbf{s}) = E(b - \mathbf{a} \cdot \mathbf{s})$. Once this computation is done, it remains to homomorphically extract the most significant bit of $b - \mathbf{a} \cdot \mathbf{s}$ as an LWE ciphertext. We summarize our requirements under the following definition. Notice that the definition makes use of two (typically different) encryption schemes:

- a scheme E , which is used internally by the accumulator, and it is left unspecified to allow a wider range of possible implementations, and
- a target encryption scheme, which for simplicity we fix to $\text{LWE}^{t/q}$ as required by our application.

The definition is easily generalized to make it parametric also with respect to the target encryption scheme.

Definition 1 (Homomorphic Accumulator). *A Homomorphic Accumulator Scheme is a quadruple of algorithms $(E, \text{Init}, \text{Incr}, \text{msbExtract})$ together with moduli t, q , where E and msbExtract may require key material related to an LWE key \mathbf{s} . For brevity, we write $\text{ACC} \leftarrow v$ for $\text{ACC} \leftarrow \text{Init}(v)$, and $\text{ACC} \stackrel{+}{\leftarrow} E(v)$ for $\text{ACC} \leftarrow \text{Incr}(\text{ACC}, E(v))$. For any $v_0, v_1 \dots v_\ell \in \mathbb{Z}_q$, after the sequence of operations*

$$\text{ACC} \leftarrow v_0; \quad \text{for } i = 1 \text{ to } \ell \text{ do } \text{ACC} \stackrel{+}{\leftarrow} E(v_i)$$

we say that we say that ACC is an ℓ -encryption of v , where $v = \sum v_i \bmod q$.

A Homomorphic Accumulator Scheme is said \mathcal{E} -correct for some function \mathcal{E} if, for any ℓ -encryption ACC of v , computing $\mathbf{c} \leftarrow \text{msbExtract}(\text{ACC})$ ensures $\mathbf{c} \in \text{LWE}_{\mathbf{s}}^{t/q}(v, \mathcal{E}(\ell))$ with overwhelming probability.

In order to use the accumulator in our refreshing function, we set $t = 4$ and we will need $\mathcal{E}(\ell) \leq q/16$. Note that the correctness requirement assumes that all ciphertexts added to the accumulator are freshly generated and independent. (In particular, although the internal encryption scheme E may enjoy useful homomorphic properties, the ciphertexts $E(v_i)$ are generated by a direct application of the encryption function E on v_i , rather than performing homomorphic operations on ciphertexts.)

Using this accumulator data structure, we describe a family of refreshing procedures (exhibiting different space/time trade-offs) parametrized by an integer B_r . (The subscript in B_r stands for Refresh, and it is used to distinguish B_r from similar basis parameters used elsewhere in the paper.) The refreshing procedure takes as input a ciphertext $(\mathbf{a}, b) \in \text{LWE}_{\mathbf{s}}^{2/q}(m, q/4)$ and a refreshing key \mathcal{K} consisting of the encryptions $K_{i,c,j} = E(cs_i B_r^j \bmod q)$ for $c \in \{0, \dots, B_r - 1\}$, $j = 0, \dots, d_r - 1$ (where $d_r = \lceil \log_{B_r} q \rceil$) and $i = 1, \dots, n$. (In total, $n B_r d_r \approx n(B_r / \log B_r) \log q$ ciphertexts). It then proceeds as described in Algorithm 1.

Algorithm 1. $\text{Refresh}_{\mathcal{K}}(\mathbf{a}, b)$, for $\mathcal{K} = \{K_{i,c,j}\}_{i \leq n, c \leq B_r, j \leq d_r}$

```

 $\text{ACC} \leftarrow b + (q/4)$ 
 $\text{for } i = 1, \dots, n \text{ do}$ 
    Compute the base- $B_r$  representation of  $-a_i = \sum_j B_r^j \cdot a_{i,j} \pmod{q}$ 
     $\text{for } j = 0, \dots, d_r - 1 \text{ do } \text{ACC} \stackrel{+}{\leftarrow} K_{i,a_{i,j},j}$ 
 $\text{end for}$ 
Output  $\text{msbExtract}(\text{ACC})$ .

```

Theorem 8. *If $(E, \text{Init}, \text{Incr}, \text{msbExtract})$ is a correct Homomorphic Accumulator Scheme, then the Refresh procedure, on input any ciphertext $\mathbf{c} \in \text{LWE}_{\mathbf{s}}^{2/q}(m, q/4)$, and a valid refreshing key $\mathcal{K} = \{K_{i,c,j} = E(cs_i B_r^j)\}_{i,c,j}$, outputs a ciphertext $\text{Refresh}_{\mathcal{K}}(\mathbf{c}) \in \text{LWE}_{\mathbf{s}}^{t/q}(m, \mathcal{E}(nd))$.*

Proof. The refreshing procedure initializes the accumulator to $b + q/4$, and then adds nd (distinct, freshly generated) ciphertexts $K_{i,a_{i,j},j} = E(a_{i,j} s_i B_r^j)$ to it. So, the final output is (with overwhelming probability) an LWE encryption with error at most $\mathcal{E}(nd)$. The implicit value v of the accumulator at the end of the main loop is

$$v - \frac{q}{4} = b + \sum_{i,j} a_{i,j} s_i B_r^j = b + \sum_i s_i \sum_j B_r^j a_{i,j} = b + - \sum_i a_i s_i = \frac{q}{2}m + e$$

where e is the error of the input ciphertext (\mathbf{a}, b) . Since $|e| < q/4$ by assumption, we have $0 < e + q/4 < q/2$. It follows that $0 < v < q/2$ if $m = 0$, and $q/2 < v < q$ if $m = 1$. Therefore, $\text{msbExtract}(\text{ACC})$ produces a $\text{LWE}_{\mathbf{s}}^{q/t}(m, \mathcal{E}(nd))$ encryption as claimed.

5 Homomorphic Accumulator from Ring-GSW

In this section we show how to implement the homomorphic accumulator scheme needed by our refreshing procedure. As a reminder, the homomorphic accumulator is parametrized by a modulus $q = 2^k$ (which we assume to be a power of 2), an integer t (in our main application, $t = 4$), and an encryption scheme E with message space \mathbb{Z}_q .

Our construction follows the suggestion of Alperin-Sheriff and Peikert [2] to generalize their scheme. Essentially, we avoid the costly construction of the additive group \mathbb{Z}_q as a subgroup of some symmetric group \mathfrak{S}_ℓ (represented as permutation matrices). Instead, we directly implement \mathbb{Z}_q as the multiplicative (sub)group of the roots of unity of the ring \mathcal{R} .

5.1 Description

The scheme is parametrized by a modulus Q , a dimension $N = 2^K$ such that q divides $2N$, and a base B_g . (Here the subscript in B_g stands for gadget.) For simplicity of the analysis, we will assume that $Q = B_g^{d_g}$ for some integer d_g , and that B_g is a power of 3. We use the rings $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_Q = (\mathcal{R}/Q\mathcal{R})$ (see Section 2), and an additional parameter u , which should be an invertible element of \mathbb{Z}_Q close to $Q/2t$. Since Q is a power of 3, either $\lfloor Q/2t \rfloor$ or $\lceil Q/2t \rceil$ is invertible, and we can let u be one of these two numbers, so that the distance $\delta = u - Q/2t$ is at most $|\delta| < 1$.

Messages $m \in \mathbb{Z}_q$ are encoded as roots of unity $Y^m \in \mathcal{R}$ where $Y = X^{2N/q}$. Notice that the roots of unity $\mathcal{G} = \langle X \rangle = \{1, X, \dots, X^{N-1}, -1, -X, \dots, -X^{N-1}\}$ form a cyclic group, and the message space $\mathbb{Z}_q \simeq \langle Y \rangle$ is a subgroup of $\mathcal{G} \simeq \mathbb{Z}_{2N}$. Our Homomorphic Accumulator Scheme is based on a variant of the GSW cryptosystem and works as follows:

- $E_z(m)$, on input a message m and a key $z \in \mathcal{R}$, picks $\mathbf{a} \in \mathcal{R}_Q^{2d_g}$ uniformly at random, and $\mathbf{e} \in \mathcal{R}^{2d_g} \simeq \mathbb{Z}^{2d_g N}$ with a subgaussian distribution χ of parameter ς , and outputs

$$E_z(m) = [\mathbf{a}, \mathbf{a} \cdot z + \mathbf{e}] + uY^m \mathbf{G} \quad \in \mathcal{R}_Q^{2d_g \times 2}$$

where $\mathbf{G} = (\mathbf{I}, B_g \mathbf{I}, \dots, B_g^{d_g-1} \mathbf{I}) \in \mathcal{R}_Q^{2d_g \times 2}$.

- **Init** ($\text{ACC} \leftarrow v$), on input $v \in \mathbb{Z}_q$, simply sets $\text{ACC} := uY^v \cdot \mathbf{G} \in \mathcal{R}_Q^{2d_g \times 2}$.
- **Incr** ($\text{ACC} \xleftarrow{+} \mathbf{C}$), on input the current accumulator content $\text{ACC} \in \mathcal{R}_Q^{2d_g \times 2}$ and a ciphertext $\mathbf{C} \in \mathcal{R}_Q^{2d_g \times 2}$, first computes the base- B_g decomposition of $u^{-1}\text{ACC} = \sum_{i=1}^{d_g} B_g^{i-1} \mathbf{D}_i$ (where each $\mathbf{D}_i \in \mathcal{R}^{2d_g \times 2}$ has entries with coefficients in $\{\frac{1-B_g}{2}, \dots, \frac{B_g-1}{2}\}$), and then updates the accumulator to

$$\text{ACC} := [\mathbf{D}_1, \dots, \mathbf{D}_{d_g}] \cdot \mathbf{C}.$$

An efficient algorithm for **Incr** using FFT/NTT will be detailed in Section 5.3.

- **msbExtract** (defined by Algorithm 2) uses a key-switching auxiliary input \mathfrak{K} (as defined in Section 2) and a testing vector $\mathbf{t} = -\sum_{i=0}^{q/2-1} \vec{Y^i}$. (On a first reading, the reader may want to focus on the special case where $q = 2N$, where the testing vector is just $\mathbf{t} = -(1, 1, \dots, 1)$.) The algorithm follows. The crux of matter for the extraction of the msb is that $\mathbf{t} \cdot \vec{Y^v} = -1$ if $0 \leq i < N$, and $+1$ if $N \leq i < 2N$.

Before providing a detailed analysis (Theorem 10) we explain the ideas behind the definitions of our homomorphic accumulator. As already mentioned, the scheme is based on a variant of the (private-key, ring-based) GSW encryption scheme. There are two main differences between our scheme and the original GSW scheme: the use of the extra parameter u (which plays an important role in our msb extraction algorithm), and the fact that the messages are encrypted in the exponent (of Y). At any point in time, the accumulator data structure holds the encryption $E_z(v)$ of some value $v \in \mathbb{Z}_q$ under a fixed key z . The initialization step **Init** simply sets **ACC** to a trivial (noiseless) encryption of v . The increment procedure is similar to the *multiplicative* homomorphism of the GSW scheme [22]. Since our messages are *in the exponent*, this provides homomorphic additions of ciphertexts.

Algorithm 2. $\text{msbExtract}_{\mathfrak{K}}(\text{ACC})$, for $\mathfrak{K} = \{\mathbf{k}_{i,j,w}\}_{i \leq N, j \leq B_{\text{ks}}, w \leq d_{\text{ks}}}$

Require: A switching key $\mathfrak{K} = \{\mathbf{k}_{i,j,w}\}_{i,j,w}$ from \mathbf{z} to \mathbf{s} : $\mathbf{k}_{i,j,w} \leftarrow \text{LWE}_{\mathbf{s}}^{q/q}(w \cdot z_i \cdot d_{\text{ks}}^j)$.

An accumulator **ACC** that is an ℓ -encryption of v .

- 1: $[\mathbf{a}^t, \mathbf{b}^t] \leftarrow ([\vec{0}^t, \mathbf{t}^t, \vec{0}^t, \dots, \vec{0}^t] \cdot \overset{\Rightarrow}{\text{ACC}}) \in \mathbb{Z}_Q^{2N} \quad // \overset{\Rightarrow}{\text{ACC}} \in \mathbb{Z}^{2Nd_g \times 2N}$
 - 2: $\mathbf{c} \leftarrow (\mathbf{a}, b_0 + u) \quad \in \text{LWE}_{\mathbf{z}}^{t/Q}(\text{msb}(v))$
 - 3: $\mathbf{c}' \leftarrow \text{KeySwitch}(\mathbf{c}, \mathfrak{K}) \quad \in \text{LWE}_{\mathbf{s}}^{t/Q}(\text{msb}(v))$
 - 4: $\mathbf{c}'' \leftarrow \text{ModSwitch}(\mathbf{c}') \quad \in \text{LWE}_{\mathbf{s}}^{t/q}(\text{msb}(v))$
 - 5: Return \mathbf{c}' .
-

5.2 Correctness

In this subsection we prove that ACC is a correct Homomorphic Accumulator Scheme for an appropriate error function \mathcal{E} . The main result is given in Theorem 10. But first, let us detail the behaviour of individual operations.

The `Init` operation $\text{ACC} \leftarrow v$ sets up ACC to a noiseless encryption of v under E_z for any secret key z . The homomorphic property of `Incr` follows from the following claim.

Fact 9. *For any messages $m, m' \in \mathbb{Z}_q$, if $\text{ACC} = [\mathbf{a}, \mathbf{a} \cdot z + \mathbf{e}] + uY^m \mathbf{G}$ and $\mathbf{C} = [\mathbf{a}', \mathbf{a}' \cdot z + \mathbf{e}'] + uY^{m'} \mathbf{G}$, then $\text{ACC} \stackrel{+}{\leftarrow} \mathbf{C}$ has the form $[\mathbf{a}'', \mathbf{a}'' \cdot z + \mathbf{e}''] + uY^{m+m'} \mathbf{G}$ for $\mathbf{e}'' = \mathbf{e} + [\mathbf{C}_1, \dots, \mathbf{C}_{d_g}] \cdot \mathbf{e}'$.*

The last operation `msbExtract` is slightly more intricate. Let us put aside the key and modulus switching steps, and consider, as in the algorithmic definition of `msbExtract`, the vector

$$[\mathbf{a}^t, b_0] \leftarrow \mathbf{t}^t \cdot [\vec{\vec{a}}, \vec{b}']$$

where $[a, b'] \in \mathcal{R}^{1 \times 2}$ is the second row of the accumulator $\text{ACC} \in \mathcal{R}^{2d_g \times 2}$. If ACC is a GSW encryption of a value v , $[a, b']$ verifies $\vec{b}' = \vec{\vec{a}} \cdot \vec{z} + u \cdot \vec{Y^v} + \mathbf{e}$ for some small error \mathbf{e} . Let's write $\vec{Y^v}$ as the vector $\mathbf{x}_{v \cdot 2N/q} \in \mathbb{Z}_Q^N$ defined as follows:

$$\mathbf{x}_i = \vec{X^i} = (\underbrace{0, \dots, 0}_{i-1}, 1, 0 \dots, 0) \text{ if } i \in \{0 \dots N-1\}, \quad \mathbf{x}_i = -\mathbf{x}_{i-N} \text{ otherwise.}$$

For $i \in \mathbb{Z}_{2N}$, summing all coordinates of \mathbf{x}_i results in $(-1)^{\text{msb}(i)}$, and $\mathbf{t}^t \cdot \vec{Y^v} = -(-1)^{\text{msb}(v)}$ for any $v \in \mathbb{Z}_q$. It remains to recall the identity $1 - (-1)^x = 2x$ for any bit $x \in \{0, 1\}$ to rewrite

$$\mathbf{c} = (\mathbf{a}, b_0 + u) = (\mathbf{a}, \mathbf{a} \cdot \vec{z} + \mathbf{t} \cdot \mathbf{e} + 2u \text{msb}(v)) \quad \text{where } \mathbf{a} = \mathbf{t}^t \cdot \vec{\vec{a}},$$

which is an $\text{LWE}_{\vec{z}}^{t/Q}$ encryption of $\text{msb}(v)$ since $u \approx Q/2t$. We may now move to the formal correctness statement, including bound on error size.

Theorem 10. *Assuming the hardness $\text{Ring-LWE}_{\mathcal{R}, Q, \chi}$ the above Homomorphic Accumulator Scheme is \mathcal{E} -correct with error function*

$$\mathcal{E}(\ell) = \sqrt{\frac{q^2}{Q^2} \left(\varsigma^2 B_{\mathbf{g}}^2 \cdot \ell \cdot q \cdot Nd_{\mathbf{g}} + \sigma^2 Nd_{\mathbf{ks}} \right) + \|\mathbf{s}\|^2 \cdot \omega(\sqrt{\log n})}.$$

We obtain Theorem 10 by combining the following Lemma 11 with the correctness of Key Switching and Modulus Switching, Lemmata 6 and 5. The hardness assumption is not strictly necessary for correctness, but does simplify the proof by allowing one to assume that fresh ciphertexts $\mathbf{C} \leftarrow E_z(\cdot)$ behave as independent uniform random matrices.

Lemma 11 (Intermediate error). *Assume the hardness of Ring-LWE_{R,Q,X}, and let ACC is an ℓ -encryption of v where $\ell \geq \omega(\sqrt{\log N})$. Then the ciphertext $\mathbf{c} \in \text{LWE}_z^{t/Q}(\text{msb}(v))$ as define in line 2 of algorithm 2 while computing $\text{msbExtract}(\text{ACC})$ has an error $\text{err}(\mathbf{c})$ which is a subgaussian with variable parameter β and mean 2δ under the randomness used in the calls to $E_z(\cdot)$, for $\beta = O(\varsigma B \sqrt{q \cdot Nd_g \cdot \ell})$.*

Let us start with the following fact.

Fact 12 (Spectral Norm of Decomposed Matrices). *Let $\mathbf{C}^{(i)} \leftarrow E_z(v^{(i)})$ be fresh encryptions of $v^{(i)} \in \mathbb{Z}_q$ for all $i \leq \ell = \omega(\sqrt{\log n})$, and assume that the $\mathbf{C}^{(i)}$'s are indistinguishable from random without the knowledge of z . Consider $\text{ACC}^{(\ell)}$ as the value of ACC after the sequence of operations:*

$$\text{ACC} \leftarrow v^{(0)}; \quad \text{for } i = 1 \dots \ell \text{ do } \text{ACC} \xleftarrow{+} \mathbf{C}^{(i)}.$$

Set $\mathbf{D}^{(i)} = [\mathbf{D}_1 \dots \mathbf{D}_{d_g}]$ to be the decomposition of $u^{-1}\text{ACC}^{(i)} = \sum_{j=1}^{d_g} B_g^{j-1} \mathbf{D}_j$. Then, with overwhelming probability we have

$$s_1\left([\mathbf{D}^{(0)}, \mathbf{D}^{(1)}, \dots, \mathbf{D}^{(\ell-1)}]\right) = O(B_g \sqrt{Nd_g \cdot \ell}).$$

Proof. Because the spectral norm $s_1([\mathbf{D}^{(0)}, \mathbf{D}^{(1)}, \dots, \mathbf{D}^{(\ell-1)}])$ is efficiently computable from the $\mathbf{C}^{(i)}$'s, we can assume without loss of generality that the $\mathbf{C}^{(i)}$'s are truly uniformly random. We prove by induction on ℓ that

1. for $1 \leq i \leq \ell$, the $\mathbf{D}^{(i)}$'s follow independent uniform distributions in $\mathcal{R}_{[B_g]}^{2d_g \times 2d_g}$ where $\mathcal{R}_{[B_g]}$ is the set of polynomials with coefficients in $\left\{\frac{1-B_g}{2}, \dots, \frac{B_g-1}{2}\right\}$.
2. for $0 \leq i \leq \ell$, $\mathbf{D}^{(i)}$ is invertible with overwhelming probability.

The implication 1. \Rightarrow 2. follows from Lemma 4. Indeed the uniform distribution over $\mathcal{R}_{[B_g]}$ is still a uniform distribution when taken mod 3 since 3 divides B_g . Note that $\mathbf{D}^{(0)} = Y^{v_0} \cdot \mathbf{I}_{2D}$ is invertible.

We may now start the induction and assume that $\mathbf{D}^{(\ell-1)}$ is invertible. It follows that $\text{ACC}^{(\ell)} = \mathbf{D}^{(\ell-1)} \cdot \mathbf{C}^{(\ell)}$ is uniformly random in $\mathcal{R}_Q^{2d_g \times D}$ and independent of all $\mathbf{D}^{(i)}$ for $i < \ell$. We conclude the induction using the fact that the decomposition step is a bijective map $\mathcal{R}_Q^{2d_g \times 2} \rightarrow \mathcal{R}_{[B_g]}^{2d_g \times 2d_g}$.

The coefficients of $\mathbf{D} = [\mathbf{D}^{(1)}, \dots, \mathbf{D}^{(\ell-1)}] \in \mathcal{R}^{2d_g \times 2d_g \ell}$ are independent subgaussian variables with parameter $O(B_g)$. It follows by lemma 1 that

$$s_1\left([\mathbf{D}^{(0)}, \mathbf{D}^{(1)}, \dots, \mathbf{D}^{(\ell-1)}]\right) \leq O(B_g \sqrt{Nd_g \cdot \ell}).$$

Proof (of Lemma 11). Applying ℓ times Fact 9, we can show that $\text{ACC}^{(\ell)}$ has the form

$$\text{ACC}^{(\ell)} = [\mathbf{A}, \mathbf{A} \cdot z + \mathbf{e}] + uX^v \mathbf{G} \quad \text{with } \mathbf{e} = \sum_{i=1}^{\ell} \mathbf{D}^{(i-1)} \mathbf{e}^{(i)}$$

where $\mathbf{e}^{(i)}$ is the error used in the encryption $\mathbf{C}^{(i)} \leftarrow E_z(v^{(i)})$. The final error in \mathbf{c} is $e = [\overrightarrow{0}^t, \mathbf{t}^t, \overrightarrow{0}^t, \dots, \overrightarrow{0}^t] \cdot \overrightarrow{\mathbf{e}}$. We rewrite

$$e = [\overrightarrow{0}^t, \mathbf{t}^t, \overrightarrow{0}^t, \dots, \overrightarrow{0}^t] \cdot \left[\overset{\Rightarrow}{\mathbf{D}^{(0)}, \dots, \mathbf{D}^{(\ell-1)}} \right] \cdot \overrightarrow{(\mathbf{e}^{(1)}, \dots, \mathbf{e}^{(\ell)})}.$$

Recall that $\|\mathbf{t}\| = \sqrt{q/2}$, and by Fact 12, we have that $[\mathbf{D}^{(0)}, \dots, \overset{\Rightarrow}{\mathbf{D}^{(\ell-1)}}]$ has spectral norm $O(B_g \sqrt{Nd_g \cdot \ell})$. We can rewrite $e = \mathbf{v} \cdot \overrightarrow{(\mathbf{e}^{(1)}, \dots, \mathbf{e}^{(\ell)})}$ where $\|\mathbf{v}\| = O(B_g \sqrt{q \cdot Nd_g \cdot \ell})$ and $(\mathbf{e}^{(1)}, \dots, \mathbf{e}^{(\ell)})$ is a subgaussian vector of parameter ς . We conclude that the final error is subgaussian of parameter $\beta = O(\varsigma B_g \sqrt{q N d_g \cdot \ell})$.

5.3 Efficient Accumulator Increment

To efficiently implement the accumulator increment `Incr`, one needs to keep the accumulator `ACC`, as well as the precomputed ciphertexts from the bootstrapping key, in FFT/NTT format.

Algorithm 3. `Incr`($\widehat{\text{ACC}} \in \widehat{\mathcal{R}}^{2d_g \times 2}, \widehat{\mathbf{C}} \in \widehat{\mathcal{R}}^{2d_g \times 2}$)

Compute $\text{ACC} \leftarrow \text{FFT}^{-1}(\widehat{\text{ACC}})$

Decompose $u^{-1} \text{ACC} = \sum_{i=1}^{d_g} B_g^{i-1} \mathbf{D}_i$, and set $\mathbf{D} = [\mathbf{D}_1 \dots \mathbf{D}_{d_g}] \in \mathcal{R}^{2d_g \times 2d_g}$

Compute $\widehat{\mathbf{D}} \leftarrow \text{FFT}(\mathbf{D})$

Return $\widehat{\mathbf{D}} \odot \widehat{\mathbf{C}}$

Each increment requires $4d_g$ backward FFT's and $4d_g^2$ forward FFT's. If one uses the Number Theoretic Transform rather than the complex FFT, $4d_g$ forward transforms can be traded for a few additions mod Q by computing $\widehat{\mathbf{D}}_1 = u^{-1} \widehat{\text{ACC}} - \sum_{i=2}^{d_g} B_g^{i-1} \cdot \widehat{\mathbf{D}}_i \pmod{Q}$.

5.4 Asymptotic Parameters and Efficiency

Secret Keys and Errors. We choose the secret key \mathbf{s} of the LWE scheme to be binary in order to minimize the final error parameter $\mathcal{E}(nd)$ that depends on $\|\mathbf{s}\|$ (Theorem 10). The hardness of LWE for such a distribution of secrets was established in [7]. The randomized rounding used for errors in the switching key $\mathbf{k}_{i,j,v} \leftarrow \text{LWE}_{\mathbf{s}}^{q/q}(v \cdot z_i \cdot d_{\mathbf{k}s}^j)$, is $\chi_\sigma(x) = D_{\mathbb{Z},x,\sigma}$, the discrete gaussian of standard deviation σ centered in x .

The secret $z \in \mathcal{R}$ of the Ring-GSW scheme follows the discrete gaussian distribution $\chi_\varsigma(0)$, and the errors follow the gaussian randomized rounding function χ_ς .

Parameters. For simplicity, we take the base $B_g, B_r, B_{ks} = \Theta(1)$ to be fixed, which sets $d_g, d_r, d_{ks} = O(\log n)$ provided that $q, Q = \text{poly}(n)$. Error parameters are set to $\sigma, \varsigma = \omega(\sqrt{\log n})$. For the dimension of the Ring-GSW scheme, we take $2N = q = \Theta(n)$. It remains to set $Q = n^2 \cdot \log n \cdot \omega(\log n)$, and we obtain a refreshing error $\mathcal{E}(nd) = O(n) \leq q/16$.

Efficiency and Comparison. The running time of the Refresh operation is dominated by dn homomorphic operations. For comparison, the scheme of [2] requires $dn \cdot O(\log^3 q / \log \log q)$ homomorphic operations.

In practice this polylogarithmic is far from negligible, e.g. $q = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 = 2310$ gives a factor $2^2 + 3^2 + 5^2 + 7^2 + 11^2 = 208$. Memory usage is also decreased by a factor $O(\log^2 q / \log \log q)$, that is a factor 28 in our previous example.

Also, we do not rely on randomized decomposition for the increment operation $\text{ACC} \xleftarrow{+} \mathbf{C}$. While this randomization is asymptotically less expensive than the FFT step by a factor $\log N$, avoiding it makes the implementation simpler and potentially faster considering the cost of randomness in practice.

Finally, our Refresh procedure (before key and modulus switching) produces a ciphertext with subgaussian error of parameter $\alpha = O(n^2 \log n)$ in our scheme against $\alpha = \Theta(n^{5/2} \log^3 n / \log \log n)$ in [2].

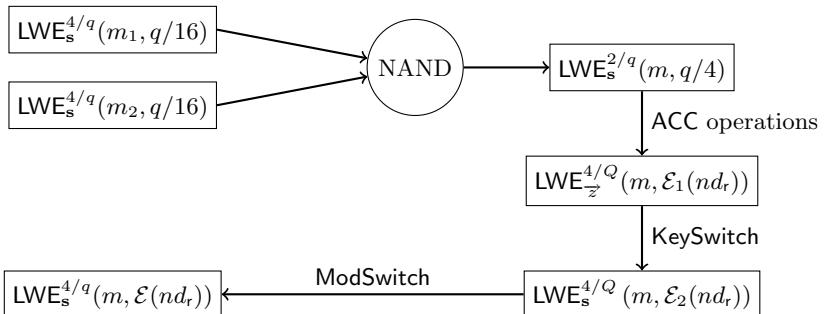


Fig. 1. Cycle for a simple NAND gate, using the Homomorphic property of Section 3

6 Parameters, Implementation and Benchmark

We start by presenting the methodology to evaluate the security of our scheme in Section 6.1, propose parameters in Section 6.2, discuss FFT implementation details in Section 6.3 and conclude with the benchmarks in Section 6.4.

6.1 Security Estimation

The security estimate methodology follows the analysis of [27]. To build an ϵ -distinguisher against LWE in dimension n , modulus q and a randomized rounding

function χ of standard deviation σ , Lindner and Peikert estimate that the best known attack by lattice reduction requires to achieve a *root Hermite factor* of

$$\delta = \delta\text{-LWE}(n, q, \sigma, \epsilon) = 2^{(\log_2^2 \rho)/(4n \log_2 q)} \quad \text{where } \rho = (q/\sigma) \cdot \sqrt{2 \ln(1/\epsilon)} \quad (5)$$

To estimate the security of $\text{binLWE}_{n,q,\sigma}$, going through the security reduction of [7] would be a very pessimistic approach. Still, $\text{binLWE}_{n,q,\sigma}$ doesn't enjoy as much concrete security as $\text{LWE}_{n,q,\sigma}$. Indeed, binary secrets allow an attacker to switch to a smaller modulus q' without affecting the relative error $1/\rho$ much (which is actually the property we exploit for the correctness of our scheme). Indeed, switching from modulus q to q' , one obtains essentially binLWE samples with errors parameter $\sigma' = \sqrt{(q'/q)^2 \sigma^2 + \|\mathbf{s}\|^2/12} \approx \sigma q'/q$, following Lemma 5. For comparison, such modulus switch on usual LWE produces errors of parameter $\sigma' = \sqrt{(q'/q)^2 \sigma^2 + \sigma^2 O(n)} \approx \sigma \sqrt{n}$.

In light of this attack, we compute the root Hermite factor for binLWE as follows:

$$\delta\text{-binLWE}(n, q, \sigma, \epsilon) = \min_{q' \leq q} \delta\text{-LWE}(n, q', \sigma') = \sqrt{(q'/q)^2 \sigma^2 + n/24}, \epsilon. \quad (6)$$

Such minimum will be computed using standard numerical analysis tools for the security estimation of our set of parameters below.

6.2 Proposed Parameters

Relaxed Constraints on B_g and Q . In practice we will ignore the constraints of the correctness statement (Theorem 10) that B_g is a power of 3 and Q is a power of B_g . Those constraints are artifact of our proofs, we will only require that $B_g^{d_g} \geq Q$. We have verified that in practice this relaxation does not significantly affects the distribution of $\text{err}(\text{Refresh}(\mathbf{c}))$.

Accumulated Errors. According to the central limit heuristic, the final error $\text{err}(\text{Refresh}(\mathbf{c}))$ of a refreshed ciphertext behaves as a Gaussian of standard deviation:

$$\beta = \sqrt{\frac{q^2}{Q^2} \left(\varsigma^2 \cdot \frac{B_r^2}{12} \cdot nd_r \cdot \frac{q}{2} \cdot 2Nd' + \sigma^2 Nd_{ks} \right) + \frac{\|\mathbf{s}\|^2 + 1}{12}}.$$

The factors $\frac{1}{12}$ follows from the fact that a uniform random variable in $[-\frac{1}{2}, \frac{1}{2}]$ has variance $\frac{1}{12}$.

The factor $2d'$ (instead of $2d_g$) takes account that the final coordinate of a decomposition of an element mod Q over base B_g is bounded by $Q/2B_g^{d_g}$ rather than $B_g/2$. Therefore we set $d' = B_g - 1 + Q/B_g^{d_g}$ (in the following parameters we have $d' = 2.5$ instead of $d_g = 3$).

Additionally, we assume that $\|\mathbf{s}\| \leq n/2$, which is true for half of the random secrets $\mathbf{s} \in \{0, 1\}^n$. If not, one may simply discard this \mathbf{s} during key generation and resample a fresh secret key. To thwart an attack that would shift all coordinates of \mathbf{s} by $-1/2$, we also randomize the signs of each entry of \mathbf{s} , which intuitively, can only increase the security (and does not affect the error analysis).

We evaluate the error probability as the probability that two independently refreshed ciphertexts $\mathbf{c}_1, \mathbf{c}_2$ verify $|\text{err}(\mathbf{c}_1) + \text{err}(\mathbf{c}_2)| < q/8$, which is sufficient but looser than $|\text{err}(\mathbf{c}_i)| < q/16$.

Parameters.

LWE parameters:	$n = 500$	$Q = 2^{32}$	$\sigma = 2^{17}$	$q = 2^9$
Ring-GSW parameters:	$N = 2^{10}$	$\varsigma = 1.4$		
Gadget Matrix:	$B_g = 2^{11}$	$d_g = 3$	$u = \frac{Q}{8} + 1$	
Bootstrapping Key parameters:	$B_r = 23$	$d_r = 2$		
Key Switching Key parameters:	$B_{ks} = 24$	$d_{ks} = 7$		

Efficiency.

Bootstrapping Key Size: $4nNd_rB_rd_g \log_2 Q$ bits = 1032 MBytes.

Key Switching Key Size: $nNB_{ks}d_{ks} \log_2 Q$ bits = 314 MBytes.

FFTs per NAND gate: $4nd_rd_g(d_g + 1)$ = 48,000 FFTs.

Correctness.

Final error parameter: $\beta = 6.94$.

Pr. of error per NAND: $p = 1 - \text{erf}(r/\sqrt{2}) \leq 2^{-31}$ where $r = \frac{q/8}{\sqrt{2}\beta}$.

The error probability can be brought down to 2^{-45} by applying the `HomNAND` operation *before* `KeySwitch` and `ModSwitch`.

Security.

Security of the LWE scheme $\delta\text{-binLWE}(n, Q, \sigma, 2^{-64}) = 1.0064$.

Security of the Ring-GSW scheme $\delta\text{-LWE}(N, Q, \varsigma, 2^{-64}) = 1.0064$.

The security of the Ring-GSW scheme is evaluated ignoring the ring structure, since there are yet no known algorithms that exploit such structure.

According to the predictions of [10], the BKZ algorithm requires a block size greater than 190 to reach a root hermite factor of 1.0065. For such block size, each of the many calls to the enumeration routine would visit more than 2^{100} nodes of the pruned enumeration tree. This is to be considered as a preliminary security analysis, demonstrating the feasibility of our construction. For a more precise security analysis, one should include the more involved results of Liu and Nguyen [28], and any new advances on lattice cryptanalysis.

6.3 FFT Implementation

To avoid implementation technicalities related to working in a prime field \mathbb{F}_Q and potentially expensive reduction mod Q , we choose to rely on the complex

FFT rather than the Number Theoretic Transform, that is, we use the complex primitive $2N$ -th root of unity $\omega = \exp(2\pi i/2N)$ rather than a primitive root in \mathbb{F}_Q . This allows us to rely on a flexible and optimized library for FFT, namely, *the Fastest Fourier Transform in the West* [13] and choose Q as a power of two, essentially offering reductions mod Q for free.

Technically, one wishes to compute the so-called negacyclic-FFT of rank N , which can be extracted from the FFT in rank $2N$ by only keeping the odd indexes of the result. Nevertheless, a factor 2 is saved considering that we are computing FFT on real-data.

Due to vectorized instructions, this implementation of FFT at double-precision reaches up to 6 Gflops on a single 64-bits Intel Core running at 3 Ghz. We measure a running time of 10 microseconds per FFT at dimension $2N = 2048$; which fits the predictions². While it is unclear if either the choice of FFT over NTT is optimal, or if this particular implementation is, this prototype is enough to support our claim.

Precision Issues. One crucial question when using complex FFT is the precision requirement. In our case (see Section 5.3), FFT is used to multiply two integer polynomials with coefficients in \mathbb{Z}_Q , yet one of them is guaranteed to have coefficients smaller than $B_g/2$. Without reduction mod Q , the resulting product is expected to have coefficients of size $S = B_g Q \sqrt{N}/4$. The final result is guaranteed to be correct if the final relative error ϵ verifies $S\epsilon \leq 1/2$. For our set of parameters, we have $S = 2^{46}$.

Asymptotically, the relative error growth during FFT is known to be $O(\log N)$ in the worst case and $O(\sqrt{\log N})$ on average [14, 33]. In practice, at double precision ($\epsilon_0 = 2^{-54}$ relative error for each operation) FFTW [13] in rank $2N = 2048$ is reported³ to produce errors of standard deviation $\epsilon = 2^{-52}$ (which match $\approx \epsilon_0 \cdot \sqrt{\log N}$). It seems barely sufficient to ensure perfect correctness of each computation of a products of polynomials. Yet, if small errors are introduced by floating-point approximations, this doesn't necessary breaks the correctness of the scheme. Indeed, this errors can simply be considered as a small extra error term introduced at each operation on the accumulator.

A formal claim would require a more detailed study. The fact that our implementation works in practice, and that the measurements of errors fit our prediction is sufficient for our purpose.

6.4 Benchmark and Source Code

Our implementation performs a HomNAND and a Refresh operation every 0.69 seconds on a single 64-bits Intel core at 3GHz, which conforms to our prediction of 0.5 seconds from the count of FFT operations (the key switching step is having non negligible cost because it hasn't been vectorized yet). It consumes 2.2Gbytes of memory, which is approximately twice the prediction. This is explained by the

² <http://www.fftw.org/speed/>

³ <http://www.fftw.org/accuracy/>

fact that for efficiency, the Bootstrapping Key is stored in FFT form, at double precision.

We can expect those performance figures to be improved by further implementation efforts. Yet, our prototype implementation already performs within one order of magnitude of the *amortized* cost of bootstrapping in HElib [23]. A more precise comparison is hard to state considering our scheme has a different security parameters, and does not offers the same set of gates. Sophisticated benchmarking would not be very useful until this new scheme is optimized and generalized to reach its full potential.

The source code is reasonably concise and simple, consisting of about 600 lines of C++ code, excluding the library FFTW. It is available on [github](#) [11].

7 Extensions, Conclusions and Future Work

We have shown that a complete bootstrappable homomorphic computation can be performed in a fraction of a second, much faster than any previous solution. We achieved the result by addressing the simplest form of bootstrappable computation (the computation of a single binary gate that is complete for boolean circuits), and introducing new techniques for this homomorphic computation. We remark that the techniques presented in the paper are not limited to NAND gates. For example, it is immediate to extend our solution to compute a majority gate that on input 3 bits $x_1, x_2, x_3 \in \{0, 1\}$, outputs 1 if at least two of the inputs are 1, and 0 if at least two of the inputs are zero. To see this, recall that our solution to the NAND problem resorted to viewing bits are integers modulo $t = 4$, and then encoding the NAND operation in terms of addition. Still using arithmetic modulo 4, one can compute the majority of x_1, x_2, x_3 by taking the sum $y = x_1 + x_2 + x_3 \in \{0, 1, 2, 3\}$, and checking if the result is at least 2. The final test is easily performed by applying our most significant bit extraction procedure to the shifted sum $y - 0.5$. As we are adding three input ciphertexts, this may require slightly smaller noise, but the computation is almost identical to the NAND gate described in this paper.

This can be further generalized to (weigthed) threshold gates $\sum_i w_i x_i > h$, where the number of inputs, weighths w_i and the threshold h are arbitrary, by using arithmetic modulo a larger $t > 2 \sum |w_i|$.

Further generalizations are possible by replacing our `msbExtract` procedure with a more complex test that checks membership for many subsets of \mathbb{Z}_t . Precisely, membership test may be extended to any anti-symmetric set $\mathcal{S} \subset \mathbb{Z}_t$ ($x \in \mathcal{S} \Leftrightarrow x + \frac{t}{2} \notin \mathcal{S}$). For example, with $t = 6$ arbitrary large xor's $x_1 \oplus \dots \oplus x_k$ can be performed in just one `Refresh` operation using the membership test $x_1 + \dots + x_k \bmod 6 \in \{1, 3, 5\}$. With this generalization, our technique also offers *xor-for-almost-free*, as in previous FHE schemes.

Additionally, taking weighted linear combinations of k input bits $\sum_i 2^i x_i$, and checking membership in subsets of $\mathbb{Z}_{2^{k+2}}$, one can (at least in principle) implement arbitrary boolean gates (adders, S-boxes, etc.), but the complexity grows exponentially in the number of inputs k .

We also remark that since the membership test is much less expensive than the rest of the Refresh procedure, one may test several function of the same input for almost free. In other words, gates with several outputs would not be much more expensive than gates with only one output. For $t = 6$, this already allows to perform an *add-with-carry* gate (3 inputs, 2 outputs) in a single shot (instead of 5 using binary gates).

Fully exploring the use of our techniques to realize more complex gates is left to future work. Other interesting open problems are finding ways to fully exploit the message space offered by ring LWE encryption in our accumulator implementation, and combining our framework with the CRT techniques of [2].

Acknowledgments. The authors wish to thank Igors Stepanovs for interesting conversations about circular security that lead us to the new homomorphic NAND procedure, as well as the anonymous EUROCRYPT’15 reviewers for their careful reading of the paper and their diligent comments.

References

1. Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 1–20. Springer, Heidelberg (2013)
2. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 297–314. Springer, Heidelberg (2014)
3. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009)
4. Blum, A., Furst, M.L., Kearns, M., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (1994)
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012)
6. Brakerski, Z., Gentry, C., Halevi, S.: Packed ciphertexts in LWE-based homomorphic encryption. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 1–13. Springer, Heidelberg (2013)
7. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Raghavendra, T., Feigenbaum, J., (eds.), 45th ACM STOC, pp. 575–584. ACM Press, June 2013
8. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 97–106. IEEE Computer Society Press, October 2011
9. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011)
10. 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)

11. Ducas, L., Micciancio, D.: Implementation of FHEW (2014). <https://github.com/lducas/FHEW>
12. Ducas, L., Micciancio, D.: Improved short lattice signatures in the standard model. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 335–352. Springer, Heidelberg (2014)
13. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proceedings of the IEEE **93**(2), 216–231 (2005). Special issue on “Program Generation, Optimization, and Platform Adaptation”
14. Gentleman, W.M., Sande, G.: Fast fourier transforms: For fun and profit. In: Proceedings of the November 7–10, 1966, Fall Joint Computer Conference, AFIPS 1966 (Fall), pp. 563–578. ACM, New York, NY, USA (1966)
15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 169–178. ACM Press, May / June 2009
16. Gentry, C., Halevi, S.: Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 107–109. IEEE Computer Society Press, October 2011
17. Gentry, C., Halevi, S.: Implementing Gentry’s fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011)
18. Gentry, C., Halevi, S., Peikert, C., Smart, N.P.: Ring switching in BGV-style homomorphic encryption. In: Visconti, I., De Prisco, R. (eds.) SCN 2012. LNCS, vol. 7485, pp. 19–37. Springer, Heidelberg (2012)
19. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 1–16. Springer, Heidelberg (2012)
20. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (2012)
21. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012)
22. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013)
23. Halevi, S., Shoup, V.: Algorithms in HElib. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 554–571. Springer, Heidelberg (2014)
24. Halevi, S., Shoup, V.: Bootstrapping for HElib. IACR Cryptology ePrint Archive, (2014). <http://eprint.iacr.org/2014/873>
25. 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)
26. Lidl, R., Niederreiter, H.: Finite Fields. Reading, MA: Addison-Wesley. Encyclopedia of Mathematics and its Applications **20** (1983)
27. 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)
28. Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: An update. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 293–309. Springer, Heidelberg (2013)

29. 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)
30. 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)
31. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In: 43rd FOCS, pp. 356–365. IEEE Computer Society Press, November 2002
32. 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, May 2005
33. Schatzman, J.C.: Accuracy of the discrete fourier transform and the fast fourier transform. SIAM J. Sci. Comput. **17**(5), 1150–1166 (1996)

Bootstrapping for `HElib`

Shai Halevi¹(✉) and Victor Shoup^{1,2}

¹ IBM Research, Yorktown Heights, NY, USA

² New York University, New York, NY, USA
shaih@alum.mit.edu

Abstract. Gentry’s bootstrapping technique is still the only known method of obtaining fully homomorphic encryption where the system’s parameters do not depend on the complexity of the evaluated functions. Bootstrapping involves a *recryption* procedure where the scheme’s decryption algorithm is evaluated homomorphically. So far, there have been precious few implementations of recryption, and fewer still that can handle “packed ciphertexts” that encrypt vectors of elements.

In the current work, we report on an implementation of recryption of fully-packed ciphertexts using the `HElib` library for somewhat-homomorphic encryption. This implementation required extending the recryption algorithms from the literature, as well as many aspects of the `HElib` library. Our implementation supports bootstrapping of packed ciphertexts over many extension fields/rings. One example that we tested involves ciphertexts that encrypt vectors of 1024 elements from $\text{GF}(2^{16})$. In that setting, the recryption procedure takes under 5.5 minutes (at security-level ≈ 76) on a single core, and allows a depth-9 computation before the next recryption is needed.

1 Introduction

Homomorphic Encryption (HE) [11, 26] enables computation of arbitrary functions on encrypted data without knowing the secret key. All current HE schemes follow Gentry’s outline from [11], where fresh ciphertexts are “noisy” to ensure security and this noise grows with every operation until it overwhelms the signal and causes decryption errors. This yields a “somewhat homomorphic” scheme (SWHE) that can only evaluate low-depth circuits, which can then be converted to a “fully homomorphic” scheme (FHE) using bootstrapping. Gentry described a *recryption* operation, where the decryption procedure of the scheme is run homomorphically, using an encryption of the secret key that can be found in the public key, resulting in a new ciphertext that encrypts the same plaintext but has smaller noise.

The last few years saw a large body of work improving many aspects of homomorphic encryption in general (e.g., [3–5, 7, 14, 17, 21]) and recryption in particular [1, 2, 10, 16, 24]. However, so far, only a few implementations of SWHE have been reported, and even fewer support recryption. Prior to the current work, we are aware of only three reported implementations of recryption: the

implementation by Gentry and Halevi [12] of Gentry’s original cryptosystem [11], the implementation of Coron, Lepoint, and Tibouchi [6–8] of van Dijk, Gentry, Halevi and Vaikuntanathan’s (DGHV) scheme over the integers [9], and the implementation by Rohloff and Cousins [27] of the NTRU-based cryptosystem [20, 21].

In this paper we report on our new implementation of recryption for the cryptosystem of Brakerski, Gentry and Vaikuntanathan (BGV) [4]. We implemented recryption on top of the open-source library **HElib** [18, 19], which implements the ring-LWE variant of BGV. Our implementation includes both new algorithmic designs as well as re-engineering of some aspects of **HElib**. As noted in [18], the choice of homomorphic primitives in **HElib** was guided to a large extent by the desire to support recryption, but nonetheless in the course of our implementation we had to extend the implementation of some of these primitives (e.g., matrix-vector multiplication), and also implement a few new ones (e.g., polynomial evaluation).

The **HElib** library is “focused on effective use of the Smart-Vercauteren ciphertext packing techniques [29] and the Gentry-Halevi-Smart optimizations [14],” so in particular we implemented recryption for “fully-packed” ciphertexts. Specifically, our implementation supports recryption of ciphertexts that encrypt vectors of elements from extension fields (or rings). Importantly, our recryption procedure itself has rather low depth (usually 10–13 levels), so as to allow significant processing between recryptions while keeping the lattice dimension reasonable to maintain efficiency.

Our results are described in Section 6; some example settings include: encrypting vectors of 1024 elements from $GF(2^{16})$ with a security level of 76 bits, where recryption takes 320 seconds and depth 12 (and allows additional computations of depth 9 between recryptions); and encrypting vectors of 960 elements from $GF(2^{24})$ with a security level of 123 bits, where recryption takes 7–8 minutes and depth 13 (and allows additional computations of depth 10 between recryptions).¹

Compared to the previous recrypt implementations, ours offers several advantages in both flexibility and speed. While the Gentry-Halevi and Rohloff-Cousins implementations only encrypt *one bit* per ciphertext, and the Coron-Lepoint-Tibouchi implementation allows *many bits* to be packed into a ciphertext, our implementation supports packed ciphertexts that encrypt vectors from the more general extension fields (and rings) already supported by **HElib**. Some examples that we tested include vectors over the fields $GF(2^{16})$, $GF(2^{25})$, $GF(2^{24})$, $GF(2^{36})$, $GF(17^{40})$, and $GF(127^{36})$, as well as degree-21 and degree-30 extensions of the ring \mathbb{Z}_{256} .

In terms of speed, the Gentry-Halevi implementation reportedly took 1/2-hour on a single core to recrypt a single-bit ciphertext. The Rohloff-Cousins implementation reported recryption time of 275 seconds on 20 cores for a single-bit ciphertext with 64-bit security. The Coron-Lepoint-Tibouchi implementation reports a recryption time of 172 seconds on 4 cores for a ciphertext with 513

¹ We used the latter setting with our re-implementation of homomorphic AES, see the long version of [15].

one-bit slots at a security level of 72. For similar setting, we clocked a single-core recryption time of 320 seconds for a ciphertext with 1024 slots of elements of $\text{GF}(2^{16})$ at a security level of 76. We note that the same parallelism that was used by Rohloff and Cousins could in principle be applied to our implementation too, but doing so involves several software-engineering challenges, and we have not yet attempted it.

Concurrent work. Concurrently with our work, Ducas and Micciancio described a new bootstrapping procedure [10]. This procedure is applied to Regev-like ciphertexts [25] that encrypt a single bit, using a secret key encrypted similarly to the new cryptosystem of Gentry et al. [17]. They reported on an implementation of their scheme, where they can perform a NAND operation followed by recryption in less than a second. Compared to our scheme, theirs has the advantage of a much faster wall-clock time for recryption, but they do not support batching or large plaintext spaces (hence our implementation has much better amortized per-bit timing). It is a very interesting open problem to combine their techniques with ours, achieving a “best of both worlds” implementation.

1.1 Algorithmic Aspects

Our recryption procedure follows the high-level structure introduced by Gentry et al. [16], and uses the tensor decomposition of Alperin-Sheriff and Peikert [1] for the linear transformations. However, those two works only dealt with characteristic-2 plaintext spaces so we had to extend some of their algorithmic components to deal with characteristics $p > 2$ (see Section 5)

Also, to get an efficient implementation, we had to make the decomposition from [1] explicit, specialize it to cases that support very-small-depth circuits, and align the different representations to reduce the required data-movement and multiplication-by-constant operations. These aspects are described in Section 4. One significant difference between our implementation and the procedure of Alperin-Sheriff and Peikert [1] is that we *do not use* the ring-switching techniques of Gentry et al. [13] (see discussion in Appendix B).

Organization. We describe our notations and give some background information on the BGV cryptosystem and the `HElib` library in Section 2. In Section 3 we provide an overview of the high-level recryption procedure from [16] and our variant of it. We then describe in detail our implementation of the linear transformations in Section 4 and the non-linear parts in Section 5. In Section 5.4 we explain how all these parts are put together in our implementation, and in Section 6 we discuss our performance results. We conclude with directions for future work in Section 7. In Appendix A we describe our choice of parameters.

2 Notations and Background

For integer z , we denote by $[z]_q$ the reduction of z modulo q into the interval $[-q/2, q/2]$, except that for $q = 2$ we reduce to $(-1, 1]$. This notation extends

to vectors and matrices coordinate-wise, and to elements of other algebraic groups/rings/fields by reducing their coefficients in some convenient basis.

For an integer z (positive or negative) we consider the base- p representation of z and denote its digits by $z\langle 0 \rangle_p, z\langle 1 \rangle_p, \dots$. When p is clear from the context we omit the subscript and just write $z\langle 0 \rangle, z\langle 1 \rangle, \dots$. When $p = 2$ we consider a 2's-complement representation of signed integers (i.e., the top bit represent a large negative number). For an odd p we consider balanced mod- p representation where all the digits are in $[-\frac{p-1}{2}, \frac{p-1}{2}]$.

For indexes $0 \leq i \leq j$ we also denote by $z\langle j, \dots, i \rangle_p$ the integer whose base- p expansion is $z\langle j \rangle \cdots z\langle i \rangle$ (with $z\langle i \rangle$ the least significant digit). Namely, for odd p we have $z\langle j, \dots, i \rangle_p = \sum_{k=i}^j z\langle k \rangle p^{k-i}$, and for $p = 2$ we have $z\langle j, \dots, i \rangle_2 = (\sum_{k=i}^{j-1} z\langle k \rangle 2^{k-i}) - z\langle j \rangle 2^{j-i}$. The properties of these representations that we use in our procedures are the following:

- For any $r \geq 1$ and any integer z we have $z = z\langle r-1, \dots, 0 \rangle \pmod{p^r}$.
- If the representation of z is d_{r-1}, \dots, d_0 then the representation of $z \cdot p^r$ is $d_{r-1}, \dots, d_0, 0, \dots, 0$ (with r zeros at the end).
- If p is odd and $|z| < p^e/2$ then the digits in positions e and up in the representation of z are all zero.
- If $p = 2$ and $|z| < 2^{e-1}$, then the bits in positions $e-1$ and up in the representation of z , are either all zero if $z \geq 0$ or all one if $z < 0$.

2.1 The BGV Cryptosystem

The BGV ring-LWE-based somewhat-homomorphic scheme [4] is defined over a ring $R \stackrel{\text{def}}{=} \mathbb{Z}[X]/(\Phi_m(X))$, where $\Phi_m(X)$ is the m th cyclotomic polynomial. For an arbitrary integer modulus N (not necessarily prime) we denote the ring $R_N \stackrel{\text{def}}{=} R/NR$. We often identify elements in R (or R_N) with their representation in some convenient basis, e.g., their coefficient vectors as polynomials. When dealing with R_N , we assume that the coefficients are in $[-N/2, N/2]$ (except for R_2 where the coefficients are in $\{0, 1\}$). We discuss these representations in some more detail in Section 4.1. The norm of an element $\|a\|$ is defined as its norm in some convenient basis.²

As implemented in **HElib**, the native plaintext space of the BGV cryptosystem is R_{p^r} for a prime power p^r . The scheme is parametrized by a sequence of decreasing moduli $q_L \gg q_{L-1} \gg \dots \gg q_0$, and an “ith level ciphertext” in the scheme is a vector $\text{ct} \in (R_{q_i})^2$. Secret keys are elements $\mathbf{s} \in R$ with “small” coefficients (chosen in $\{0, \pm 1\}$ in **HElib**), and we view \mathbf{s} as the second element of the 2-vector $\text{sk} = (1, \mathbf{s}) \in R^2$. A level- i ciphertext $\text{ct} = (\mathbf{c}_0, \mathbf{c}_1)$ encrypts a plaintext element $\mathbf{m} \in R_{p^r}$ with respect to $\text{sk} = (1, \mathbf{s})$ if we have $[\langle \text{sk}, \text{ct} \rangle]_{q_i} = [\mathbf{c}_0 + \mathbf{s} \cdot \mathbf{c}_1]_{q_i} = \mathbf{m} + p^r \cdot \mathbf{e}$ (in R) for some “small” error term, $p^r \cdot \|\mathbf{e}\| \ll q_i$.

² The difference between the norm in the different bases is not very important for the current work.

The error term grows with homomorphic operations of the cryptosystem, and switching from q_{i+1} to q_i is used to decrease the error term roughly by the ratio q_{i+1}/q_i . Once we have a level-0 ciphertext ct , we can no longer use that technique to reduce the noise. To enable further computation, we need to use Gentry’s bootstrapping technique [11], whereby we “recrypt” the ciphertext ct , to obtain a new ciphertext ct^* that encrypts the same element of R_{p^r} with respect to some level $i > 0$.

In **HElib**, each q_i is a product of small (machine-word sized) primes. Elements of the ring R_{q_i} are typically represented in *DoubleCRT* format: as a vector of polynomials modulo each small prime t , each of which itself is represented by its evaluation at the primitive m th roots of unity in \mathbb{Z}_t . In DoubleCRT format, elements of R_{q_i} may be added and multiplied in linear time. Conversion between DoubleCRT representation and the more natural coefficient representation may be affected in quasi-linear time using the FFT.

2.2 Encoding Vectors in Plaintext Slots

As observed by Smart and Vercauteren [29], an element of the native plaintext space $\alpha \in R_{p^r}$ can be viewed as encoding a vector of “plaintext slots” containing elements from some smaller ring extension of \mathbb{Z}_{p^r} via Chinese remaindering. In this way, a single arithmetic operation on α corresponds to the same operation applied component-wise to all the slots.

Specifically, suppose the factorization of $\Phi_m(X)$ modulo p^r is $\Phi_m(X) \equiv F_1(X) \cdots F_k(X) \pmod{p^r}$, where each F_i has the same degree d , which is equal to the order of p modulo m . (This factorization can be obtained by factoring $\Phi_m(X)$ modulo p and then Hensel lifting.) From the CRT for polynomials, we have the isomorphism

$$R_{p^r} \cong \bigoplus_{i=1}^k (\mathbb{Z}[X]/(p^r, F_i(X))).$$

Let us now define $E \stackrel{\text{def}}{=} \mathbb{Z}[X]/(p^r, F_1(X))$, and let ζ be the residue class of X in E , which is a principal m th root of unity, so that $E = \mathbb{Z}/(p^r)[\zeta]$. The rings $\mathbb{Z}[X]/(p^r, F_i(X))$ for $i = 1, \dots, k$ are all isomorphic to E , and their direct product is isomorphic to R_{p^r} , so we get an isomorphism between R_{p^r} and E^k . **HElib** makes extensive use of this isomorphism, representing it explicitly as follows. It maintains a set $S \subset \mathbb{Z}$ that forms a complete system of representatives for the quotient group $\mathbb{Z}_m^*/\langle p \rangle$, i.e., it contains exactly one element from every residue class. Then we use a ring isomorphism

$$R_{p^r} \rightarrow \bigoplus_{h \in S} E, \quad \alpha \mapsto \{\alpha(\zeta^h)\}_{h \in S}. \tag{1}$$

Here, if α is the residue class $a(X) + (p^r, \Phi_m(X))$ for some $a(X) \in \mathbb{Z}[X]$, then $\alpha(\zeta^h) = a(\zeta^h) \in E$, which is independent of the representative $a(X)$.

This representation allows **HElib** to effectively pack $k \stackrel{\text{def}}{=} |S| = |\mathbb{Z}_m^*/\langle p \rangle|$ elements of E into different “slots” of a single plaintext. Addition and multiplication of ciphertexts act on the slots of the corresponding plaintext in parallel.

2.3 Hypercube Structure and One-Dimensional Rotations

Beyond addition and multiplications, we can also manipulate elements in R_{p^r} using a set of automorphisms on R_{p^r} of the form $a(X) \mapsto a(X^j)$, or in more detail

$$\tau_j : R_{p^r} \rightarrow R_{p^r}, \quad a(X) + (p^r, \Phi_m(X)) \mapsto a(X^j) + (p^r, \Phi_m(X)) \quad (j \in \mathbb{Z}_m^*).$$

We can homomorphically apply these automorphisms by applying them to the ciphertext elements and then performing “key switching” (see [4, 14]). As discussed in [14], these automorphisms induce a hypercube structure on the plaintext slots, where the hypercube structure depends on the structure of the group $\mathbb{Z}_m^*/\langle p \rangle$. Specifically, **HElib** keeps a hypercube basis $g_1, \dots, g_n \in \mathbb{Z}_m$ with orders $\ell_1, \dots, \ell_n \in \mathbb{Z}_{>0}$, and then defines the set S of representatives for $\mathbb{Z}_m^*/\langle p \rangle$ (which is used for slot mapping Eqn. (1)) as

$$S \stackrel{\text{def}}{=} \{g_1^{e_1} \cdots g_n^{e_n} \bmod m : 0 \leq e_i < \ell_i, i = 1, \dots, n\}. \quad (2)$$

This basis defines an n -dimensional hypercube structure on the plaintext slots, where slots are indexed by tuples (e_1, \dots, e_n) with $0 \leq e_i < \ell_i$. If we fix $e_1, \dots, e_{i-1}, e_{i+1}, \dots, e_n$, and let e_i range over $0, \dots, \ell_i - 1$, we get a set of ℓ_i slots, indexed by (e_1, \dots, e_n) , which we refer to as a *hypercolumn in dimension i* (and there are k/ℓ_i such hypercolumns). Using automorphisms, we can efficiently perform rotations in any dimension; a rotation by v in dimension i maps a slot indexed by $(e_1, \dots, e_i, \dots, e_n)$ to the slot indexed by $(e_1, \dots, e_i + v \bmod \ell_i, \dots, e_n)$. Below we denote this operation by ρ_i^v .

We can implement ρ_i^v by applying either one automorphism or two: if the order of g_i in \mathbb{Z}_m^* is ℓ_i , then we get by with just a single automorphism, $\rho_i^v(\alpha) = \tau_{g_i^v}(\alpha)$. If the order of g_i in \mathbb{Z}_m^* is different from ℓ_i then we need to implement this rotation using two shifts: specifically, we use a constant “0-1 mask value” **mask** that selects some slots and zeros-out the others, and use two automorphisms with exponents $e = g_i^v \bmod m$ and $e' = g_i^{v-\ell_i} \bmod m$, setting $\rho_i^v(\alpha) = \tau_e(\text{mask} \cdot \alpha) + \tau_{e'}((1 - \text{mask}) \cdot \alpha)$. In the first case (where one automorphism suffices) we call i a “good dimension”, and otherwise we call i a “bad dimension”.

2.4 Frobenius and Linearized Polynomials

We define $\sigma \stackrel{\text{def}}{=} \tau_p$, which is the Frobenius map on R_{p^r} . It acts on each slot independently as the Frobenius map σ_E on E , which sends ζ to ζ^p and leaves elements of \mathbb{Z}_{p^r} fixed. (When $r = 1$, σ is the same as the p th power map on E .) For any \mathbb{Z}_{p^r} -linear transformation on E , denoted M , there exist unique constants $\theta_0, \dots, \theta_{d-1} \in E$ such that $M(\eta) = \sum_{f=0}^{d-1} \theta_f \sigma_E^f(\eta)$ for all $\eta \in E$. When

$r = 1$, this follows from the general theory of linearized polynomials (see, e.g., Theorem 10.4.4 on p. 237 of [28]), and these constants are readily computable by solving a system of equations mod p ; when $r > 1$, we may Hensel-lift these mod- p solutions to a solution mod p^r . In the special case where the image of M is the sub-ring \mathbb{Z}_{p^r} of E , the constants θ_f are obtained as $\theta_f = \sigma_E^f(\theta_0)$ for $f = 1, \dots, d - 1$; again, this is standard field theory if $r = 1$, and is easily established for $r > 1$ as well.

Using linearized polynomials, we may effectively apply a fixed linear map to each slot of a plaintext element $\alpha \in R_{p^r}$ (either the same or different maps in each slot) by computing $\sum_{f=0}^{d-1} \kappa_f \sigma^f(\alpha)$, where the κ_f 's are R_{p^r} -constants obtained by embedding appropriate E -constants in the slots. Applying such linear \mathbb{Z}_{p^r} -linear transformation(s) homomorphically on an encryption of α takes $d - 1$ automorphisms and d constant-ciphertext multiplications, and can be done in depth of one constant-ciphertext multiplication (since automorphisms consume almost no depth).

3 Overview of the Recryption Procedure

Recall that the recryption procedure is given a BGV ciphertext $\text{ct} = (\mathbf{c}_0, \mathbf{c}_1)$, defined relative to secret-key $\text{sk} = (1, \mathfrak{s})$, modulus q , and plaintext space p^r , namely, we have $[\langle \text{sk}, \text{ct} \rangle]_q \equiv \mathbf{m} \pmod{p^r}$ with \mathbf{m} being the plaintext. Also we have the guarantee that the noise is ct is still rather small, say $\|[\langle \text{sk}, \text{ct} \rangle]_q\| < q/100$.

The goal of the recryption procedure is to produce another ciphertext ct^* that encrypts the same plaintext element \mathbf{m} relative to the same secret key, but relative to a much larger modulus $Q \gg q$ and with a much smaller relative noise. That is, we still want to get $[\langle \text{sk}, \text{ct}^* \rangle]_Q = \mathbf{m} \pmod{p^r}$, but with $\|[\langle \text{sk}, \text{ct}^* \rangle]_Q\| \ll Q$.³ Our implementation uses roughly the same high-level structure for the recryption procedure as in [1, 16], below we briefly recall the structure from [16] and then describe our variant of it.

3.1 The GHS Recryption Procedure

The recryption procedure from [16] (for plaintext space $p = 2$) begins by using modulus-switching to compute another ciphertext that encrypts the same plaintext as ct , but relative to a specially chosen modulus $\tilde{q} = 2^e + 1$ (for some integer e).

Denote the resulting ciphertext by ct' , the rest of the recryption procedure consists of homomorphic implementation of the decryption formula $\mathbf{m} \leftarrow [[\langle \text{sk}, \text{ct}' \rangle]_{\tilde{q}}]_2$, applied to an encryption of sk that can be found in the public key. Note that in this formula we know $\text{ct}' = (\mathbf{c}'_0, \mathbf{c}'_1)$ explicitly, and it is sk that we

³ The relative noise after recryption is a design parameter. In our implementation we tried to get the noise below $Q/2^{250}$, to allow significant additional processing before another recryption is needed.

process homomorphically. It was shown in [16] that for the special modulus \tilde{q} , the decryption procedure can be evaluated (roughly) by computing $u \leftarrow [\langle sk, ct' \rangle]_{2^{e+1}}$ and then $m \leftarrow u\langle e \rangle \oplus u\langle 0 \rangle$.⁴

To enable recryption, the public key is augmented with an encryption of the secret key s , relative to a (much) larger modulus $Q \gg \tilde{q}$, and also relative to a larger plaintext space 2^{e+1} . Namely this is a ciphertext \tilde{ct} such that $[\langle sk, \tilde{ct} \rangle]_Q = s \pmod{2^{e+1}}$. Recalling that all the coefficients in $ct' = (c'_0, c'_1)$ are smaller than $\tilde{q}/2 < 2^{e+1}/2$, we consider c'_0, c'_1 as plaintext elements modulo 2^{e+1} , and compute homomorphically the inner-product $u \leftarrow c'_1 \cdot s + c'_0 \pmod{2^{e+1}}$ by setting $\tilde{ct}' \leftarrow c'_1 \cdot \tilde{ct} + (c'_0, 0)$. This means that \tilde{ct}' encrypts the desired u , and to complete the recryption procedure we just need to extract and XOR the top and bottom bits from all the coefficients in u , thus getting an encryption of (the coefficients of) the plaintext m . This calculation is the most expensive part of recryption, and it is done in three steps:

Linear transformation. First apply homomorphically a $Z_{2^{e+1}}$ -linear transformation to \tilde{ct}' , converting it into ciphertexts that have the coefficients of u in the plaintext slots.

Bit extraction. Next apply a homomorphic (non-linear) bit-extraction procedure, computing two ciphertexts that contain the top and bottom bits (respectively) of the integers stored in the slots. A side-effect of the bit-extraction computation is that the plaintext space is reduced from mod- 2^{e+1} to mod-2, so adding the two ciphertexts we get a ciphertext whose slots contain the coefficients of m relative to a mod-2 plaintext space.

Inverse linear transformation. Finally apply homomorphically the inverse linear transformation (this time over Z_2), obtaining a ciphertext ct^* that encrypts the plaintext element m .

An optimization. The deepest part of recryption is bit-extraction, and its complexity — both time and depth — increases with the most-significant extracted bit (i.e., with e). The parameter e can be made somewhat smaller by choosing a smaller $\tilde{q} = 2^e + 1$, but for various reasons \tilde{q} cannot be too small, so Gentry et al. described in [16] an optimization for reducing the top extracted bit without reducing \tilde{q} .

After modulus-switching to the ciphertext ct , we can add multiples of \tilde{q} to the coefficients of c'_0, c'_1 to make them divisible by $2^{e'}$ for some moderate-size $e' < e$. Let $ct'' = (c''_0, c''_1)$ be the resulting ciphertext, clearly $[\langle sk, ct' \rangle]_{\tilde{q}} = [\langle sk, ct'' \rangle]_{\tilde{q}}$ so ct'' still encrypts the same plaintext m . Moreover, as long as the coefficients of ct'' are sufficiently smaller than \tilde{q}^2 , we can still use the same simplified decryption formula $u' \leftarrow [\langle sk, ct'' \rangle]_{2^{e+1}}$ and $m \leftarrow u'\langle e \rangle \oplus u'\langle 0 \rangle$.

However, since ct'' is divisible by $2^{e'}$ then so is u' . For one thing this means that $u'\langle 0 \rangle = 0$ so the decryption procedure can be simplified to $m \leftarrow u'\langle e \rangle$.

⁴ This is a slight simplification, the actual formula for $p = 2$ is $m \leftarrow u\langle e \rangle \oplus u\langle e - 1 \rangle \oplus u\langle 0 \rangle$, see Lemma 2.

But more importantly, we can divide ct'' by $2^{e'}$ and compute instead $\mathbf{u}'' \leftarrow [\langle \mathbf{sk}, \text{ct}''/2^{e'} \rangle]_{2^{e-e'+1}}$ and $\mathbf{m} \leftarrow \mathbf{u}' \langle e - e' \rangle$. This means that the encryption of \mathbf{s} in the public key can be done relative to plaintext space $2^{e-e'}$ and we only need to extract $e - e'$ bits rather than e .

In this work we observe that we can do even slightly better by adding to ct' multiples of \tilde{q} and also multiples of 2 (or more generally multiples of \tilde{q} and p when recrypting a ciphertext with mod- p plaintext space). This lets us get a value of e' which is one larger than what we can get by adding only multiples of \tilde{q} , so we can extract one less digit. See details in Section 5.2.

3.2 Our Recryption Procedure

We optimize the GHS recryption procedure and extend it to handle plaintext spaces modulo arbitrary prime powers p^r rather than just $p = 2$, $r = 1$. The high-level structure of the procedure remains roughly the same.

To reduce the complexity as much as we can, we use a special recryption key $\tilde{\mathbf{sk}} = (1, \tilde{\mathbf{s}})$, which is chosen as sparse as possible (subject to security requirements). As we elaborate in Appendix A, the number of nonzero coefficients in $\tilde{\mathbf{s}}$ plays an extremely important role in the complexity of recryption.⁵

To enable recryption of mod- p^r ciphertexts, we include in the public key a ciphertext $\tilde{\mathbf{ct}}$ that encrypts the secret key $\tilde{\mathbf{s}}$ relative to a large modulus Q and plaintext space mod- p^{e+r} for some $e > r$. Then given a mod- p^r ciphertext \mathbf{ct} to recrypt, we perform the following steps:

Modulus-switching. Convert \mathbf{ct} into another \mathbf{ct}' relative to the special modulus $\tilde{q} = p^e + 1$. We prove in Lemma 2 that for the special modulus \tilde{q} , the decryption procedure can be evaluated by computing $\mathbf{u} \leftarrow [\langle \mathbf{sk}, \mathbf{ct}' \rangle]_{p^{e+r}}$ and then $\mathbf{m} \leftarrow \mathbf{u} \langle r - 1, \dots, 0 \rangle_p - \mathbf{u} \langle e + r - 1, \dots, e \rangle_p \pmod{p^r}$.

Optimization. Add multiples of \tilde{q} and multiples of p^r to the coefficients of \mathbf{ct}' , making them divisible by $p^{e'}$ for some $r \leq e' < e$ without increasing them too much and also without increasing the noise too much. This is described in Section 5.2. The resulting ciphertext, which is divisible by $p^{e'}$, is denoted $\mathbf{ct}'' = (\mathbf{c}_0'', \mathbf{c}_1'')$. It follows from the same reasoning as above that we can now compute $\mathbf{u}' \leftarrow [\langle \mathbf{sk}, \mathbf{ct}''/p^{e'} \rangle]_{p^{e-e'+r}}$ and then $\mathbf{m} \leftarrow -\mathbf{u}' \langle e - e' + r - 1, \dots, e - e' \rangle_p \pmod{p^r}$.

Multiply by encrypted key. Evaluate homomorphically the inner product (divided by $p^{e'}$), $\mathbf{u}' \leftarrow (\mathbf{c}_1' \cdot \mathbf{s} + \mathbf{c}_0')/p^{e'} \pmod{p^{e-e'+r}}$, by setting $\tilde{\mathbf{ct}}' \leftarrow (\mathbf{c}_1'/p^{e'}) \cdot \tilde{\mathbf{ct}} + (\mathbf{c}_0'/p^{e'}, 0)$. The plaintext space of the resulting $\tilde{\mathbf{ct}}'$ is modulo $p^{e-e'+r}$.

Note that since we only use plaintext space modulo $p^{e-e'+r}$, then we might as well use the same plaintext space also for $\tilde{\mathbf{ct}}$, rather than encrypting it relative to plaintext space modulo p^{e+r} as described above.

⁵ In our implementation we use a Hamming-weight-56 key, which is slightly smaller than the default Hamming-weight-64-keys that are used elsewhere in **HElib**.

Linear transformation. Apply homomorphically a $Z_{p^{e-e'+r}}$ -linear transformation to \tilde{ct}' , converting it into ciphertexts that have the coefficients of u' in the plaintext slots. This linear transformation, which is the most intricate part of the implementation, is described in Section 4. It uses a tensor decomposition similar to [1] to reduce complexity, but pays much closer attention to details such as the mult-by-constant depth and data movements.

Digit extraction. Apply a homomorphic (non-linear) digit-extraction procedure, computing r ciphertexts that contain the digits $e - e' + r - 1$ through $e - e'$ of the integers in the slots, respectively, relative to plaintext space mod- p^r . This requires that we generalize the bit-extraction procedure from [16] to a digit-extraction procedure for any prime power $p^r \geq 2$, this is done in Section 5.3. Once we extracted all these digits, we can combine them to get an encryption of the coefficients of m in the slots relative to plaintext space modulo p^r .

Inverse linear transformation. Finally apply homomorphically the inverse linear transformation, this time over Z_{p^r} , converting the ciphertext into an encryption ct^* of the plaintext element m itself. This too is described in Section 4.

4 The Linear Transformations

In this section we describe the linear transformations that we apply during the recryption procedure to map the plaintext coefficients into the slots and back. Central to our implementation is imposing a hypercube structure on the plaintext space $R_{p^r} = \mathbb{Z}_{p^r}[X]/(\Phi_m(X))$ with one dimension per factor of m , and implementing the second (inverse) transformation as a sequence of multi-point polynomial-evaluation operations, one for each dimension of the hypercube. We begin with some additional background.

4.1 Algebraic Background

Let m denote the parameter defining the underlying cyclotomic ring in an instance of the BGV cryptosystem with native plaintext space $R_{p^r} = \mathbb{Z}_{p^r}[X]/(\Phi_m(X))$. Throughout this section, we consider a particular factorization $m = m_1 \cdots m_t$, where the m_i 's are pairwise co-prime positive integers. We write $CRT(h_1, \dots, h_t)$ for the unique solution $h \in \{0, \dots, m - 1\}$ to the system of congruences $h \equiv h_i \pmod{m_i}$ ($i = 1, \dots, t$), where $h_i \in \{0, \dots, m_i - 1\}$ for all $i = 1, \dots, t$.

Lemma 1. *Let p, m and the m_i 's be as above, where p is a prime not dividing any of the m_i 's. Let d_1 be the order of p modulo m_1 and for $i = 2, \dots, t$ let d_i be the order of $p^{d_1 \cdots d_{i-1}}$ modulo m_i . Then the order of p modulo m is $d \stackrel{\text{def}}{=} d_1 \cdots d_t$.*

Moreover, suppose that S_1, \dots, S_t are sets of integers such that each $S_i \subseteq \{0, \dots, m_i - 1\}$ forms a complete system of representatives for $\mathbb{Z}_{m_i}^/\langle p^{d_1 \cdots d_{i-1}} \rangle$.*

Then the set $S \stackrel{\text{def}}{=} CRT(S_1, \dots, S_t)$ forms a complete system of representatives for $\mathbb{Z}_m^/\langle p \rangle$.*

Proof. It suffices to prove the lemma for $t = 2$. The general case follows by induction on t .

The fact that the order of p modulo $m \stackrel{\text{def}}{=} m_1 m_2$ is $d \stackrel{\text{def}}{=} d_1 d_2$ is clear by definition. The cardinality of S_1 is $\phi(m_1)/d_1$ and of S_2 is $\phi(m_2)/d_2$, and so the cardinality of S is $\phi(m_1)\phi(m_2)/d_1 d_2 = \phi(m)/d = |\mathbb{Z}_m^*/\langle p \rangle|$. So it suffices to show that distinct elements of S belong to distinct cosets of $\langle p \rangle$ in \mathbb{Z}_m^* .

To this end, let $a, b \in S$, and assume that $p^f a \equiv b \pmod{m}$ for some nonnegative integer f . We want to show that $a = b$. Now, since the congruence $p^f a \equiv b$ holds modulo m , it holds modulo m_1 as well, and by the defining property of S_1 and the construction of S , we must have $a \equiv b \pmod{m_1}$. So we may cancel a and b from both sides of the congruence $p^f a \equiv b \pmod{m_1}$, obtaining $p^f \equiv 1 \pmod{m_1}$, and from the defining property of d_1 , we must have $d_1 \mid f$. Again, since the congruence $p^f a \equiv b$ holds modulo m , it holds modulo m_2 as well, and since $d_1 \mid f$, by the defining property of S_2 and the construction of S , we must have $a \equiv b \pmod{m_2}$. It follows that $a \equiv b \pmod{m}$, and hence $a = b$.

The powerful basis. The linear transformations in our recryption procedure make use of the same tensor decomposition that was used by Alperin-Sheriff and Peikert in [1], which in turn relies on the “powerful basis” representation of the plaintext space, due to Lyubashevsky et al. [22, 23]. The “powerful basis” representation is an isomorphism

$$R_{p^r} = \mathbb{Z}[X]/(p^r, \Phi_m(X)) \cong R'_{p^r} \stackrel{\text{def}}{=} \mathbb{Z}[X_1, \dots, X_t]/(p^r, \Phi_{m_1}(X_1), \dots, \Phi_{m_t}(X_t)),$$

defined explicitly by the map $\text{PowToPoly} : R'_{p^r} \rightarrow R_{p^r}$ that sends (the residue class of) X_i to (the residue class of) X^{m/m_i} .

Recall that we view an element in the native plaintext space R_{p^r} as encoding a vector of plaintext slots from E , where E is an extension ring of \mathbb{Z}_{p^r} that contains a principal m th root of unity ζ . Below let us define $\zeta_i \stackrel{\text{def}}{=} \zeta^{m/m_i}$ for $i = 1, \dots, t$. It follows from the definitions above that for $h = \text{CRT}(h_1, \dots, h_t)$ and $\alpha = \text{PowToPoly}(\alpha')$, we have $\alpha(\zeta^h) = \alpha'(\zeta_1^{h_1}, \dots, \zeta_t^{h_t})$.

Using Lemma 1, we can generalize the above to multi-point evaluation. Let S_1, \dots, S_t and S be sets as defined in the lemma. Then evaluating an element $\alpha' \in R'_{p^r}$ at all points $(\zeta_1^{h_1}, \dots, \zeta_t^{h_t})$, where (h_1, \dots, h_t) ranges over $S_1 \times \dots \times S_t$, is equivalent to evaluating the corresponding element in $\alpha \in R_{p^r}$ at all points ζ^h , where h ranges over S .

4.2 The Evaluation Map

With the background above, we can now describe our implementation of the linear transformations. Recall that these transformations are needed to map the coefficients of the plaintext into the slots and back. Importantly, it is the powerful basis coefficients that we put in the slots during the first linear transformation, and take from the slots in the second transformation.

Since the two linear transformations are inverses of each other (except modulo different powers of p), then once we have an implementation of one we also get

an implementation of the other. For didactic reasons we begin by describing in detail the second transformation, and later we explain how to get from it also the implementation of the first transformation.

The second transformation begins with a plaintext element β that contains in its slots the powerful-basis coefficients of some other element α , and ends with the element α itself. Important to our implementation is the view of this transformation as *multi-point evaluation* of a polynomial. Namely, the second transformation begins with an element β whose slots contain the coefficients of the powerful basis $\alpha' = \text{PowToPoly}(\alpha)$, and ends with the element α that holds in the slots the values $\alpha(\zeta^h) = \alpha'(\zeta_1^{h_1}, \dots, \zeta_t^{h_t})$, where the h_i 's range over the S_i 's from Lemma 1 and correspondingly h range over S . Crucial to this view is that the CRT set S from Lemma 1 is the same as the representative-set S from Eqn. (2) that determines the plaintext slots.

Choosing the representatives. Our first order of business is therefore to match up the sets S from Eqn. (2) and Lemma 1. To facilitate this (and also other aspects of our implementation), we place some constraints on our choice of the parameter m and its factorization.⁶ Recall that we consider the factorization $m = m_1 \cdots m_t$, and denote by d_i the order of $p^{d_1 \cdots d_{i-1}}$ modulo m_i .

- I. In choosing m and the m_i 's we restrict ourselves to the case where each group $\mathbb{Z}_{m_i}^*/\langle p^{d_1 \cdots d_{i-1}} \rangle$ is cyclic of order k_i , and let its generator be denoted by (the residue class of) $\tilde{g}_i \in \{0, \dots, m_i - 1\}$. Then for $i = 1, \dots, t$, we set $S_i \stackrel{\text{def}}{=} \{\tilde{g}_i^e \bmod m_i : 0 \leq e < k_i\}$.

We define $g_i \stackrel{\text{def}}{=} \text{CRT}(1, \dots, 1, \tilde{g}_i, 1, \dots, 1)$ (with \tilde{g}_i in the i th position), and use the g_i 's as our hypercube basis with the order of g_i set to k_i . In this setting, the set S from Lemma 1 coincides with the set S in Eqn. (2); that is, we have $S = \{\prod_{i=1}^t g_i^{e_i} \bmod m : 0 \leq e_i < k_i\} = \text{CRT}(S_1, \dots, S_t)$.

- II. We further restrict ourselves to only use factorizations $m = m_1 \cdots m_t$ for which $d_1 = d$. (That is, the order of p is the same in $\mathbb{Z}_{m_1}^*$ as in \mathbb{Z}_m^* .) With this assumption, we have $d_2 = \cdots = d_t = 1$, and moreover $k_1 = \phi(m_1)/d$ and $k_i = \phi(m_i)$ for $i = 2, \dots, t$.

Note that with the above assumptions, the first dimension could be either good or bad, but the other dimensions $2, \dots, t$ are always good. This is because $p^{d_1 \cdots d_{i-1}} \equiv 1 \pmod{m}$, so also $p^{d_1 \cdots d_{i-1}} \equiv 1 \pmod{m_i}$, and therefore $\mathbb{Z}_{m_i}^*/\langle p^{d_1 \cdots d_{i-1}} \rangle = \mathbb{Z}_{m_i}^*$, which means that the order of g_i in \mathbb{Z}_m^* (which is the same as the order of \tilde{g}_i in $\mathbb{Z}_{m_i}^*$) equals k_i .

Packing the coefficients. In designing the linear transformation, we have the freedom to choose how we want the coefficients of α' to be packed in the slots of β . Let us denote these coefficients by c_{j_1, \dots, j_t} where each index j_i runs over

⁶ As we discuss in Section 6, there are still sufficiently many settings that satisfy these requirements.

$\{0, \dots, \phi(m_i) - 1\}$, and each c_{j_1, \dots, j_t} is in \mathbb{Z}_{p^r} . That is, we have

$$\alpha'(X_1, \dots, X_t) = \sum_{j_1, j_2, \dots, j_t} c_{j_1, \dots, j_t} X_1^{j_1} X_2^{j_2} \cdots X_t^{j_t} = \sum_{j_2, \dots, j_t} \left(\sum_{j_1} c_{j_1, \dots, j_t} X_1^{j_1} \right) X_2^{j_2} \cdots X_t^{j_t}.$$

Recall that we can pack d coefficients into a slot, so for fixed j_2, \dots, j_t , we can pack the $\phi(m_1)$ coefficients of the polynomial $\sum_{j_1} c_{j_1, \dots, j_t} X_1^{j_1}$ into $k_1 = \phi(m_1)/d$ slots. In our implementation we pack these coefficients into the slots indexed by (e_1, j_2, \dots, j_t) , for $e_1 = 0, \dots, k_1 - 1$. That is, we pack them into a single hypercolumn in dimension 1.

The Eval Transformation. The second (inverse) linear transformation of the recryption procedure begins with the element β whose slots pack the coefficients c_{j_1, \dots, j_t} as above. The desired output from this transformation is the element whose slots contain $\alpha(\zeta^h)$ for all $h \in S$ (namely the element α itself). Specifically, we need each slot of α with hypercube index (e_1, \dots, e_t) to hold the value

$$\alpha'(\zeta_1^{g_1^{e_1}}, \dots, \zeta_t^{g_t^{e_t}}) = \alpha(\zeta^{g_1^{e_1} \cdots g_t^{e_t}}).$$

Below we denote $\zeta_{i, e_i} \stackrel{\text{def}}{=} \zeta_i^{g_i^{e_i}}$. We transform β into α in t stages, each of which can be viewed as multi-point evaluation of polynomials along one dimension of the hypercube.

Stage 1. This stage begins with the element β , in which each dimension-1 hypercolumn with index (\star, j_2, \dots, j_t) contains the coefficients of the univariate polynomial $P_{j_2, \dots, j_t}(X_1) \stackrel{\text{def}}{=} \sum_{j_1} c_{j_1, \dots, j_t} X_1^{j_1}$. We transform β into β_1 where that hypercolumn contains the evaluation of the same polynomial in many points. Specifically, the slot of β_1 indexed by (e_1, j_2, \dots, j_t) contains the value $P_{j_2, \dots, j_t}(\zeta_{1, e_1})$.

By definition, this stage consists of parallel application of a particular \mathbb{Z}_{p^r} -linear transformation M_1 (namely a multi-point polynomial evaluation map) to each of the k/k_1 hypercolumns in dimension 1. In other words, M_1 maps $(k_1 \cdot d)$ -dimensional vectors over \mathbb{Z}_{p^r} (each packed into k_1 slots) to k_1 -dimensional vectors over E . We elaborate on the efficient implementation of this stage later in this section.

Stages 2, ..., t. The element β_1 from the previous stage holds in its slots the coefficients of the k_1 multivariate polynomials $A_{e_1}(\cdot)$ (for $e_1 = 0, \dots, k_1 - 1$),

$$A_{e_1}(X_2, \dots, X_t) \stackrel{\text{def}}{=} \alpha'(\zeta_{1, e_1}, X_2, \dots, X_t) = \sum_{j_2, \dots, j_t} \underbrace{\left(\sum_{j_1} c_{j_1, \dots, j_t} \zeta_{1, e_1}^{j_1} \right)}_{\text{slot } (e_1, j_2, \dots, j_t) = P_{j_2, \dots, j_t}(\zeta_{1, e_1})} \cdot X_2^{j_2} \cdots X_t^{j_t}.$$

The goal in the remaining stages is to implement multi-point evaluation of these polynomials at all the points $X_i = \zeta_{i, e_i}$ for $0 \leq e_i < k_i$. Note that differently from

the polynomial α' that we started with, the polynomials A_{e_1} have coefficients from E (rather than from \mathbb{Z}_{p^r}), and these coefficients are encoded one per slot (rather than d per slot). As we explain later, this makes it easier to implement the desired multi-point evaluation. Separating out the second dimension we can write

$$A_{e_1}(X_2, \dots, X_t) = \sum_{j_3, \dots, j_t} \left(\sum_{j_2} P_{j_2, \dots, j_t}(\zeta_{1, e_1}) X_2^{j_2} \right) X_3^{j_3} \cdots X_t^{j_t}.$$

We note that each dimension-2 hypercolumn in β_1 with index $(e_1, \star, j_3, \dots, j_t)$ contains the E -coefficients of the univariate polynomial $Q_{e_1, j_3, \dots, j_t}(X_2) \stackrel{\text{def}}{=} \sum_{j_2} P_{j_2, \dots, j_t}(\zeta_{1, e_1}) X_2^{j_2}$. In Stage 2, we transform β_1 into β_2 where that hypercolumn contains the evaluation of the same polynomial in many points. Specifically, the slot of β_2 indexed by $(e_1, e_2, j_3, \dots, j_t)$ contains the value

$$Q_{e_1, j_3, \dots, j_t}(\zeta_{2, e_2}) = \sum_{j_2} P_{j_2, \dots, j_t}(\zeta_{1, e_1}) \cdot \zeta_{2, e_2}^{j_2} = \sum_{j_1, j_2} c_{j_1, \dots, j_t} \zeta_{1, e_1}^{j_1} \zeta_{2, e_2}^{j_2},$$

and the following stages implement the multi-point evaluation of these polynomials at all the points $X_i = \zeta_{i, e_i}$ for $0 \leq e_i < k_i$.

Stages $s = 3, \dots, t$ proceed analogously to Stage 2, each time eliminating a single variable X_s via the parallel application of an E -linear map M_s to each of the k/k_s hypercolumns in dimension s . When all of these stages are completed, we have in every slot with index (e_1, \dots, e_t) the value $\alpha'(\zeta_{1, e_1}, \dots, \zeta_{t, e_t})$, as needed.

Implementing stages 2, …, t. For $s = 2, \dots, t$, we obtain β_s from β_{s-1} by applying the linear transformation M_s in parallel to each hypercolumn in dimension s . We adapt for that purpose the **HElib** matrix-multiplication procedure [18], using only rotations along dimension s . The procedure from [18] multiplies an $n \times n$ matrix M by a $n \times 1$ column vector v by computing

$$Mv = D_0 v_0 + \cdots + D_{n-1} v_{n-1}, \quad (3)$$

where each v_i is the vector obtained by rotating the entries of v by i positions, and each D_i is a diagonal matrix containing one diagonal of M . In our case, we perform k/k_s such computations in parallel, one on every hypercolumn along the s dimension, implementing the rotations using the ρ_s^e maps. That is, we set

$$\beta_s = \sum_{e=0}^{k_s-1} \kappa_{s,e} \cdot \rho_s^e(\beta_{s-1}), \quad (4)$$

where the $\kappa_{s,e}$'s are constants in R_{p^r} obtained by embedding appropriate constants in E in each slot. Eqn. (4) translates directly into a simple homomorphic evaluation algorithm, just by applying the same operations to the ciphertexts. The cost in time for stage s is $k_s - 1$ automorphisms and k_s constant-ciphertext multiplications; the cost in depth is a single constant-ciphertext multiplication.

Implementing Stage 1. Stage 1 is more challenging, because the map M_1 is a \mathbb{Z}_{p^r} -linear map, rather than an E -linear map. Nevertheless, we can still use the same diagonal decomposition as in Eqn. (3), except that the entries in the diagonal matrices are no longer elements of E , but rather, \mathbb{Z}_{p^r} -linear maps on E . These maps may be encoded using linearized polynomials, as in Section 2.4, allowing us to write

$$\beta_1 = \sum_{e=0}^{k_1-1} \sum_{f=0}^{d-1} \lambda_{e,f} \cdot \sigma^f(\rho_1^e(\beta)), \quad (5)$$

where the $\lambda_{e,f}$'s are constants in R_{p^r} .

A naive homomorphic implementation of the formula from Eqn. (5) takes $O(dk_1)$ automorphisms, but we can reduce this to $O(d + k_1)$ as follows. Since σ^f is a ring automorphism, it commutes with addition and multiplication, so we can rewrite Eqn. (5) as follows:

$$\beta_1 = \sum_{f=0}^{d-1} \sum_{e=0}^{k_1-1} \sigma^f(\sigma^{-f}(\lambda_{e,f}) \cdot \rho_1^e(\beta)) = \sum_{f=0}^{d-1} \sigma^f \left(\sum_{e=0}^{k_1-1} \sigma^{-f}(\lambda_{e,f}) \cdot \rho_1^e(\beta) \right). \quad (6)$$

To evaluate Eqn. (6) homomorphically, we compute encryptions of $\rho_1^e(\beta)$ for $e = 0, \dots, k_1 - 1$, then take d different linear combinations of these values, homomorphically computing

$$\gamma_f = \sum_{e=0}^{k_1-1} \sigma^{-f}(\lambda_{e,f}) \cdot \rho_1^e(\beta) \quad (f = 0, \dots, d-1).$$

Finally, we can compute an encryption of $\beta_1 = \sum_{f=0}^{d-1} \sigma^f(\gamma_f)$ by applying Frobenius maps to the ciphertexts encrypting the γ_f 's, and summing.

If dimension 1 is good, the homomorphic computation of the γ_f 's takes the time of $k_1 - 1$ automorphisms and $k_1 d$ constant-ciphertext multiplications, and the depth of one constant-ciphertext multiplication. If dimension 1 is bad, we can maintain the same depth by folding the multiplication by the constants $\sigma^{-f}(\lambda_{e,f})$ into the masks used for rotation (see Section 2.3); the time increases to $2(k_1 - 1)$ automorphisms and $(2k_1 - 1)d$ constant-ciphertext multiplications.

The entire procedure to compute an encryption of β_1 has depth of one constant-ciphertext multiplication, and it takes time $k_1 + d - 2 + B(k_1 - 1)$ automorphisms and $k_1 d + B(k_1 - 1)d$ constant-ciphertext multiplications, where B is a flag which is 1 if dimension 1 is bad and 0 if it is good.

Complexity of Eval. From the above, we get the following cost estimates for computing the **Eval** map homomorphically. The depth is t constant-ciphertext multiplications, and the time is at most

- $(B + 1)\phi(m_1)/d + d + \phi(m_1) + \dots + \phi(m_t)$ automorphisms, and
- $(B + 1)\phi(m_1) + \phi(m_2) + \dots + \phi(m_t)$ constant-ciphertext multiplications.

The Transformation Eval^{-1} The first linear transformation in the recryption procedure is the inverse of Eval . This transformation can be implemented by simply running the above stages in reverse order and using the inverse linear maps M_s^{-1} in place of M_s . The complexity estimates are identical.

4.3 Unpacking and Repacking the Slots

In our recryption procedure we have the non-linear digit extraction routine “sandwiched” between the linear evaluation map and its inverse. However the evaluation map transformations from above maintain fully-packed ciphertexts, where each slot contains an element of the extension ring E (of degree d), while our digit extraction routine needs “sparsely packed” slots containing only integers from \mathbb{Z}_{p^r} .

Therefore, before we can use the digit extraction procedure we need to “unpack” the slots, so as to get d ciphertexts in which each slot contains a single coefficient in the constant term. Similarly, after digit extraction we have to “repack” the slots, before running the second transformation.

Unpacking. Consider the unpacking procedure in terms of the element $\beta \in R_{p^r}$. Each slot of β contains an element of E which we write as $\sum_{i=0}^{d-1} a_i \zeta^i$ with the a_i 's in \mathbb{Z}_{p^r} . We want to compute $\beta^{(0)}, \dots, \beta^{(d-1)}$, so that the corresponding slot of each $\beta^{(i)}$ contains a_i . To obtain $\beta^{(i)}$, we need to apply to each slot of β the \mathbb{Z}_{p^r} -linear map $L_i : E \rightarrow \mathbb{Z}_{p^r}$ that maps $\sum_{i=0}^{d-1} a_i \zeta^i$ to a_i .

Using linearized polynomials, as discussed in Section 2.4, we may write $\beta^{(i)} = \sum_{f=0}^{d-1} \kappa_{i,f} \sigma^f(\beta)$, for constants $\kappa_{i,f} \in R_{p^r}$. Given an encryption of β , we can compute encryptions of all of the $\sigma^f(\beta)$'s and then take linear combinations of these to get encryptions of all of the $\beta^{(i)}$'s. This takes the time of $d - 1$ automorphisms and d^2 constant-ciphertext multiplications, and a depth of one constant-ciphertext multiplication.

While the cost in time of constant-ciphertext multiplications is relatively cheap, it cannot be ignored, especially as we have to compute d^2 of them. In our implementation, the cost is dominated the time it takes to convert an element in R_{p^r} to its corresponding DoubleCRT representation. It is possible, of course, to precompute and store all d^2 of these constants in DoubleCRT format, but the space requirement is significant: for typical parameters, our implementation takes about 4MB to store a single constant in DoubleCRT format, so for example with $d = 24$, these constants take up almost 2.5GB of space.

This unappealing space/time trade-off can be improved considerably using somewhat more sophisticated implementations. Suppose that in the first linear transformation Eval^{-1} , instead of packing the coefficients a_0, \dots, a_{d-1} into a slot as $\sum_i a_i \zeta^i$, we pack them as $\sum_i a_i \sigma_E^i(\theta)$, where $\theta \in E$ is a normal element. Further, let $L'_0 : E \rightarrow \mathbb{Z}_{p^r}$ be the \mathbb{Z}_{p^r} -linear map that sends $\eta = \sum_i a_i \sigma_E^i(\theta)$ to a_0 . Then we have $L'_0(\sigma^{-j}(\eta)) = a_j$ for $j = 0, \dots, d - 1$. If we realize the map L'_0 with linearized polynomials, and if the plaintext γ has the coefficients packed into slots via a normal element as above, then we have $\beta^{(i)} = \sum_{f=0}^{d-1} \kappa_f \cdot \sigma^{f-i}(\gamma)$, where the κ_f 's are constants in R_{p^r} . So we have only d constants rather than d^2 .

To use this strategy, however, we must address the issue of how to modify the Eval^{-1} transformation so that Eval^{-1} will give us the plaintext element γ that packs coefficients as $\sum_i a_i \sigma_E^i(\theta)$. As it turns out, in our implementation this modification is for free: recall that the unpacking transformation immediately follows the last stage of the inverse evaluation map Eval^{-1} , and that last stage applies \mathbb{Z}_{p^r} -linear maps to the slots; therefore, we simply fold into these maps the \mathbb{Z}_{p^r} -linear map that takes $\sum_i a_i \zeta^i$ to $\sum_i a_i \sigma_E^i(\theta)$ in each slot.

It is possible to reduce the number of stored constants even further: since L'_0 is a map from E to the base ring \mathbb{Z}_{p^r} , then the κ_f 's are related via $\kappa_f = \sigma^f(\kappa_0)$. Therefore, we can obtain all of the DoubleCRTs for the κ_f 's by computing just one for κ_0 and then applying the Frobenius automorphisms directly to the DoubleCRT for κ_0 . We note, however, that applying these automorphisms directly to DoubleCRTs leads to a slight increase in the noise of the homomorphic computation. We did not use this last optimization in our implementation.

Repacking. Finally, we discuss the reverse transformation, which repacks the slots, taking $\beta^{(0)}, \dots, \beta^{(d-1)}$ to β . This is quite straightforward: if $\bar{\zeta}$ is the plaintext element with ζ in each slot, then $\beta = \sum_{i=0}^{d-1} \zeta^i \beta^{(i)}$. This formula can be evaluated homomorphically with a cost in time of d constant-ciphertext multiplications, and a cost in depth one constant-ciphertext multiplication.

5 Recryption with Plaintext Space Modulo $p > 2$

Below we extend the treatment from [1, 16] to handle plaintext spaces modulo $p > 2$. In Sections 5.1 through 5.3 we generalize the various lemmas to $p > 2$, in Appendix A we discuss the choice of parameters, and then in Section 5.4 we explain how these lemmas are put together in the recryption procedure.

5.1 Simpler Decryption Formula

We begin by extending the simplified decryption formula [16, Lemma 1] from plaintext space mod-2 to any prime-power p^r . Recall that we denote by $[z]_q$ the mod- q reduction into $[-q/2, q/2)$ (except when $q = 2$ we reduce to $(-1, 1]$). Also $z\langle j, \dots, i \rangle_p$ denotes the integer whose mod- p expansion consists of digits i through j in the mod- p expansion of z (and we omit the p subscript if it is clear from the context).

Lemma 2. *Let $p > 1, r \geq 1, e \geq r + 2$ and $q = p^e + 1$ be integers, and also let z be an integer such that $|z| \leq \frac{q^2}{4} - q$ and $|[z]_q| \leq \frac{q}{4}$.*

- If p is odd then $[z]_q = z\langle r-1, \dots, 0 \rangle - z\langle e+r-1, \dots, e \rangle \pmod{p^r}$.
- If $p = 2$ then $[z]_q = z\langle r-1, \dots, 0 \rangle - z\langle e+r-1, \dots, e \rangle - z\langle e-1 \rangle \pmod{2^r}$.

Proof. We begin with the odd- p case. Denote $z_0 = [z]_q$, then $z = z_0 + kq$ for some $|k| \leq \frac{q}{4} - 1$, and hence $|z_0 + k| \leq \frac{q}{4} + \frac{q}{4} - 1 = (q-2)/2 = (p^e - 1)/2$. We can write

$$z = z_0 + kq = z_0 + k(p^e + 1) = z_0 + k + p^e k. \quad (7)$$

This means in particular that $z = z_0 + k \pmod{p^r}$, and also since the mod- p representation of the sum $w = z_0 + k$ has only 0's in positions e and up then $k\langle r-1, \dots, 0 \rangle = z\langle e+r-1, \dots, e \rangle$. It follows that

$$\begin{aligned} z_0\langle r-1, \dots, 0 \rangle &= z\langle r-1, \dots, 0 \rangle - k\langle r-1, \dots, 0 \rangle \\ &= z\langle r-1, \dots, 0 \rangle - z\langle e+r-1, \dots, e \rangle \pmod{p^r}. \end{aligned}$$

The proof for the $p = 2$ case is similar, but we no longer have the guarantee that the high-order bits of the sum $w = z_0 + k$ are all zero. Hence from Eqn. (7) we can only deduce that

$$z\langle e+r-1, \dots, e \rangle = w\langle e+r-1, \dots, e \rangle + k\langle r-1, \dots, 0 \rangle \pmod{2^r},$$

and also that $z\langle e-1 \rangle = w\langle e-1 \rangle$.

Since $|w| \leq |z_0| + |k| < \lfloor q/2 \rfloor = 2^{e-1}$, then the bits in positions $e-1$ and up in the representation of w are either all zero if $w \geq 0$, or all one if $w < 0$. In particular, this means that

$$w\langle e+r-1, \dots, e \rangle = \begin{cases} 0 & \text{if } w \geq 0 \\ -1 & \text{if } w < 0 \end{cases} = -w\langle e-1 \rangle = -z\langle e-1 \rangle \pmod{2^r}.$$

Concluding, we therefore have

$$\begin{aligned} z_0\langle r-1, \dots, 0 \rangle &= z\langle r-1, \dots, 0 \rangle - k\langle r-1, \dots, 0 \rangle \\ &= z\langle r-1, \dots, 0 \rangle - (z\langle e+r-1, \dots, e \rangle - w\langle e+r-1, \dots, e \rangle) \\ &= z\langle r-1, \dots, 0 \rangle - z\langle e+r-1, \dots, e \rangle - z\langle e-1 \rangle \pmod{2^r}. \end{aligned}$$

5.2 Making an Integer Divisible By $p^{e'}$

As sketched in Section 3, we use the following lemma to reduce to number of digits that needs to be extracted, hence reducing the time and depth of the digit-extraction step.

Lemma 3. *Let z be an integer, and let p, q, r, e' be positive integers s.t. $e' \geq r$ and $q = 1 \pmod{p^{e'}}$. Also let α be an arbitrary real number in $[0, 1]$. Then there are integer coefficients u, v such that*

$$z + u \cdot p^r + v \cdot q = 0 \pmod{p^{e'}}$$

and moreover u, v are small. Specifically $|v| \leq p^r(\frac{1}{2} + \lfloor (1-\alpha)p^{e'-1}/2 \rfloor)$ and $|u| \leq \lceil \alpha p^{e'-1}/2 \rceil$.

Proof. Since q, p are co-prime then there exists $v' \in (-p^r/2, p^r/2]$ s.t. $z' = z + v'q = 0 \pmod{p^r}$. Let $\delta = -z' \cdot p^{-r} \pmod{p^{e'-1}}$, reduced into the interval $[-\frac{p^{e'-1}}{2}, \frac{p^{e'-1}}{2}]$, so we have $|\delta| \leq p^{e'-1}/2$ and $z' + p^r\delta = 0 \pmod{p^{e'}}$. Denote $\beta = 1 - \alpha$ and consider the integer $z'' \stackrel{\text{def}}{=} z' + \lceil \alpha\delta \rceil \cdot p^r + \lfloor \beta\delta \rfloor p^r \cdot q$.

On one hand, we have that $z'' = z + u \cdot p^r + v \cdot q$ with $|u| = |\lceil \alpha\delta \rceil| \leq |\lceil \alpha p^{e'-1}/2 \rceil|$ and $|v| = |v' + p^r \lfloor \beta\delta \rfloor| \leq p^r(\frac{1}{2} + \lfloor \beta p^{e'-1}/2 \rfloor)$. On the other hand since $q = 1 \pmod{p^{e'}}$ then we also have $z'' = z' + p^r(\lceil \alpha\delta \rceil + \lfloor \beta\delta \rfloor) = z' + p^r\delta = 0 \pmod{p^{e'}}$.

Discussion. Recall that in our recryption procedure we have a ciphertext \mathbf{ct} that encrypts some \mathbf{m} with respect to modulus q and plaintext space mod- p^r , and we use the lemma above to convert it into another ciphertext \mathbf{ct}' that encrypts the same thing but is divisible by $p^{e'}$, and by doing so we need to extract e' fewer digits in the digit-extraction step.

Considering the elements $\mathbf{u} \leftarrow \langle \mathbf{sk}, \mathbf{ct} \rangle$ and $\mathbf{u}' \leftarrow \langle \mathbf{sk}, \mathbf{ct}' \rangle$ (without any modular reduction), since \mathbf{sk} is integral then adding multiples of q to the coefficients of \mathbf{ct} does not change $[\mathbf{u}]_q$, and also as long as we do not wrap around q then adding multiples of p^r does not change $[[\mathbf{u}]_q]_{p^r}$. Hence as long as we only add small multiples of p^r then we have $[[\mathbf{u}]_q]_{p^r} = [[\mathbf{u}']_q]_{p^r}$, so \mathbf{ct} and \mathbf{ct}' still encrypt the same plaintext. However in our recryption procedure we need more: to use our simpler decryption formula from Lemma 2 we not only need the noise magnitude $\|[\mathbf{u}']_q\|$ to be smaller than $\frac{q}{4}$, but the magnitude of \mathbf{u}' itself (before mod- q reduction) must be smaller than $\frac{q^2}{4} - q$.

In essence, the two types of additive terms consume two types of “resources”: adding multiples of q increases the magnitude of \mathbf{u}' , and adding multiple of p^r increases the magnitude of $[\mathbf{u}']_q$. The parameter α from Lemma 3 above lets us trade-off these two resources: smaller α means slower increase in $\|[\mathbf{u}']_q\|$ but faster increase in $\|\mathbf{u}'\|$, and vice versa for larger α . As we discuss in Appendix A, the best trade-off is often obtained when α is just under $\frac{1}{2}$; our implementation tries to optimize this parameter, and for many settings it uses $\alpha \approx 0.45$.

5.3 Digit-Extraction for Plaintext Space Modulo p^r

The bit-extraction procedure that was described by Gentry et al. in [16] and further optimized by Alperin-Sheriff and Peikert in [1] is specific for the case $p = 2^e$. Namely, for an input ciphertext relative to mod- 2^e plaintext space, encrypting some integer z (in one of the slots), this procedure computes the i th top bit of z (in the same slot), relative to plaintext space mod- 2^{e-i+1} . Below we show how to extend this bit-extraction procedure to a digit-extraction also when p is an odd prime.

The main observation underlying the original bit-extraction procedure, is that squaring an integer keeps the least-significant bit unchanged but inserts zeros in the higher-order bits. Namely, if b is the least significant bit of the integer z and moreover $z = b \pmod{2^e}$, $e \geq 1$, then squaring z we get $z^2 = b \pmod{2^{e+1}}$. Therefore, $z - z^2$ is divisible by 2^e , and the LSB of $(z - z^2)/2^e$ is the e th bit of z .

Unfortunately the same does not hold when using a base $p > 2$. Instead, we show below that for any exponent e there exists some degree- p polynomial $F_e(\cdot)$ (but not necessarily $F_e(X) = X^p$) such that when $z = z_0 \pmod{p^e}$ then $F_e(z) = z_0 \pmod{p^{e+1}}$. Hence $z - F_e(z)$ is divisible by p^e , and the least-significant digit of $(z - F_e(z))/p^e$ is the e th digit of z . The existence of such polynomial $F_e(X)$ follows from the simple derivation below.

Lemma 4. *For every prime p and exponent $e \geq 1$, and every integer z of the form $z = z_0 + p^e z_1$ (with z_0, z_1 integers, $z_0 \in [p]$), it holds that $z^p = z_0 \pmod{p}$, and $z^p = z_0^p \pmod{p^{e+1}}$.*

Proof. The first equality is obvious, and the proof of the second equality is just by the binomial expansion of $(z_0 + p^e z_1)^p$.

Corollary 1. *For every prime p there exist a sequence of integer polynomials f_1, f_2, \dots , all of degree $\leq p-1$, such that for every exponent $e \geq 1$ and every integer $z = z_0 + p^e z_1$ (with z_0, z_1 integers, $z_0 \in [p]$), we have $z^p = z_0 + \sum_{i=1}^e f_i(z_0)p^i \pmod{p^{e+1}}$.*

Proof. modulo- p^{e+1} depend only on z_0 , so there exist some polynomials in z_0 that describe them, $f_i(z_0) = z^p \langle i \rangle_p$. Since these f_i 's are polynomials from Z_p to itself, then they have degree at most $p-1$. Moreover, by the 1st equality in Lemma 4 we have that the first digit is exactly z_0 .

Corollary 2. *For every prime p and every $e \geq 1$ there exist a degree- p polynomial F_e , such that for every integers z_0, z_1 with $z_0 \in [p]$ and every $1 \leq e' \leq e$ we have $F_e(z_0 + p^{e'} z_1) = z_0 \pmod{p^{e'+1}}$.*

Proof. Denote $z = z_0 + p^{e'} z_1$. Since $z = z_0 \pmod{p^{e'}}$ then $f_i(z_0) = f_i(z) \pmod{p^{e'}}$. This implies that for all $i \geq 1$ we have $f_i(z_0)p^i = f_i(z)p^i \pmod{p^{e'+1}}$, and of course also for $i \geq e' + 1$ we have $f_i(z)p^i = 0 \pmod{p^{e'+1}}$. Therefore, setting $F_e(X) = X^p - \sum_{i=1}^e f_i(X)p^i$ we get

$$F_e(z) = z^p - \sum_{i=1}^e f_i(z)p^i = z^p - \sum_{i=1}^{e'} f_i(z_0)p^i = z_0 \pmod{p^{e'+1}}.$$

We know that for $p = 2$ we have $F_e(X) = X^2$ for all e . One can verify that also for $p = 3$ we have $F_e(X) = X^3$ for all e (when considering the balanced mod-3 representation), but for larger primes we no longer have $F_e(X) = X^p$.

The digit-extraction procedure. Just like in the base-2 case, in the procedure for extracting the e th base- p digit from the integer $z = \sum_i z_i p^i$ proceeds by computing integers $w_{j,k}$ ($k \geq j$) such that the lowest digit in $w_{j,k}$ is z_j , and the next $k-j$ digits are zeros. The code in Figure 1 is purposely written to be similar to the code from [1, Appendix B], with the only difference being in Line 5 where we use $F_e(X)$ rather than X^2 .

In our implementation we compute the coefficients of the polynomial F_e once and store them for future use. In the procedure itself, we apply a homomorphic polynomial-evaluation procedure to compute $F_e(w_{j,k})$ in Line 5. We note that just as in [1, 16], the homomorphic division-by- p operation is done by multiplying the ciphertext by the constant $p^{-1} \pmod{q}$, where q is the current modulus. Since the encrypted values are guaranteed to be divisible by p , then this has the desired effect and also it reduces the noise magnitude by a factor of p . Correctness of the procedure from Figure 1 is proved exactly the same way as in [1, 16], the proof is omitted here. In the full version we show that for $p = 2$ we can extract $r \geq 2$ consecutive bits using one level less than in the procedure above.

```

Digit-Extractionp( $z, e$ ): // Extract eth digit in base- $p$  representation of  $z$ 
1.  $w_{0,0} \leftarrow z$ 
2. For  $k = 0$  to  $e - 1$ 
3.    $y \leftarrow z$ 
4.   For  $j = 0$  to  $k$ 
5.      $w_{j,k+1} \leftarrow F_e(w_{j,k})$  //  $F_e$  from Corollary 2, for  $p = 2, 3$  we have  $F_e(X) = X^p$ 
6.      $y \leftarrow (y - w_{j,k+1})/p$ 
7.    $w_{k+1,k+1} \leftarrow y$ 
8. Return  $w_{e,e}$ 

```

Fig. 1. The digit extraction procedure

5.4 Putting Everything Together

Having described all separate parts of our recryption procedure, we now explain how they are combined in our implementation.

Initialization and parameters. Given the ring parameter m (that specifies the m th cyclotomic ring of integers $R = \mathbb{Z}[X]/(\Phi_m(X))$) and the plaintext space p^r , we compute the recryption parameters as explained in Appendix A. That is, we use compute the Hamming weight of recryption secret key $t \geq 56$, some value of α (which is often $\alpha \approx 0.45$), and some values for e, e' where $e - e' - r \in \{\lceil \log_p(t + 2) \rceil - 1, \lceil \log_p(t + 2) \rceil\}$. We also precompute some key-independent tables for use in the linear transformations, with the first transformation using plaintext space $p^{e-e'+r}$ and the second transformation using plaintext space p^r .

Key generation. During key generation we choose in addition to the “standard” secret key sk also a separate secret recryption key $\tilde{\text{sk}} = (1, \tilde{s})$, with \tilde{s} having Hamming weight t . We include in the secret key both a key-switching matrix from sk to $\tilde{\text{sk}}$, and a ciphertext $\tilde{\text{ct}}$ that encrypts \tilde{s} under key sk , relative to plaintext space $p^{e-e'+r}$.

The recryption procedure itself. When we want to recrypt a mod- p^r ciphertext ct relative to the “standard” key sk , we first key-switch it to $\tilde{\text{sk}}$ and modulus-switch it to $\tilde{q} = p^e + 1$, then make its coefficients divisible by $p^{e'}$ using the procedure from Lemma 3, thus getting a new ciphertext $\text{ct}' = (\mathbf{c}_0', \mathbf{c}_1')$. We then compute the homomorphic inner-product divided by $p^{e'}$, by setting $\text{ct}'' = (\mathbf{c}_1'/p^{e'}) \cdot \tilde{\text{ct}} + (0, \mathbf{c}_0'/p^{e'})$.

Next we apply the first linear transformation (the map Eval^{-1} from Section 4.2), moving to the slots the coefficients of the plaintext \mathbf{u}' that is encrypted in ct'' . The result is a single ciphertext with *fully packed slots*, where each slot holds d of the coefficients from \mathbf{u}' . Before we can apply the digit-extraction procedure from Section 5.3, we therefore need to *unpack* the slots, so as to put each coefficient in its own slot, which results in d “sparsely packed” ciphertexts (as described in Section 4.3).

Next we apply the digit-extraction procedure from Section 5.3 to each one of these d “sparsely packed” ciphertexts. For each one we extract the digits up

to $e + r - e'$ and combine the top digits as per Lemma 2 to get in the slots the coefficients of the plaintext polynomial \mathbf{m} (one coefficient per slot). The resulting ciphertexts all have plaintext space mod- p^r .

Next we re-combine the d ciphertext into a single fully-packed ciphertext (as described in Section 4.3) and finally apply the second linear transformation (the map Eval described in Section 4.2). This completes the recryption procedure.

6 Implementation and Performance

As discussed in Section 4.2, our algorithms for the linear transformations rely on the parameter m having a fairly special form. Luckily, there are quite a few such m 's, which we found by brute-force search. We ran a simple program that searches through a range of possible m 's (odd, not divisible by p , and not prime). For each such m , we first compute the order d of $p \bmod m$. If this exceeds a threshold (we chose a threshold of 100), we skip this m . Next, we compute the factorization of m into prime powers as $m = m_1 \cdots m_t$. We then find all indexes i such that p has order $d \bmod m_i$ and all pairs of indexes i, j such that p has order $d \bmod m_i m_j$. If we find none, we skip this m ; otherwise, we choose one such index, or pair of indexes, in such a way to balance the time and depth complexity of the linear transformations (so m_i or $m_i m_j$ becomes the new m_1 , and the other prime power factors are ordered arbitrarily).

Table 1. Experimental results with plaintext space $\text{GF}(2^d)$

cyclotomic ring m	21845 =257·5·17	18631 =601·31	28679 =241·17·7	35113 =(73·13)·37
lattice dim. $\phi(m)$	16384	18000	23040	31104
plaintext space	$\text{GF}(2^{16})$	$\text{GF}(2^{25})$	$\text{GF}(2^{24})$	$\text{GF}(2^{36})$
number of slots	1024	720	960	864
security level	76	110	96	159
before/after levels	22/10	20/10	24/11	24/12
initialization (sec)	177	248	224	694
linear transforms (sec)	127	131	123	325
digit extraction (sec)	193	293	342	1206
total recrypt (sec)	320	424	465	1531
space usage (GB)	3.4	3.5	3.5	8.2

For example, with $p = 2$, we processed all potential m 's between 16,000 and 64,000. Among these, there were a total of 377 useful m 's with $15,000 \leq \phi(m) \leq 60,016$, with a fairly even spread (the largest gap between successive $\phi(m)$'s was less than 2,500, and there were no other gaps that exceeded 2,000). So while such useful m 's are relatively rare, there are still plenty to choose from. We ran this parameter-generation program to find potential settings for plaintext-space modulo $p = 2, p = 17, p = 127$, and $p^r = 2^8$, and manually chose a few of the suggested values of m for our tests.

Table 2. Experimental results with other plaintext spaces

cyclotomic ring m	45551 =(41·11)·101	51319 =(19·73)·37	42799 = 337·127	49981 =331·151
lattice dim. $\phi(m)$	40000	46656	42336	49981
plaintext space	GF(17 ⁴⁰)	GF(127 ³⁶)	$R(256, 21)$	$R(256, 30)$
number of slots	1000	1296	2016	1650
security level	106	161	79	91
before/after levels	38/10	32/11	52/6	56/10
initialization (sec)	1148	2787	1202	1533
linear transforms (sec)	735	774	2265	2834
digit extraction (sec)	3135	1861	8542	14616
total recrypt (sec)	3870	2635	10807	17448
space usage (GB)	14.8	39.9	15.6	21.6

For each of these values of m, p, r , we then ran a test in which we chose three random keys, and performed recryption three times per key (for each key recrypting the same ciphertext over and over). These tests were run on a five-year-old IBM BladeCenter HS22/7870, with two Intel X5570 (4-core) processors, running at 2.93GHz. All of our programs are single-threaded, so only one core was used in the computations. Tables 1 and 2 summarize the results from our experiments.

In each table, the first row gives m and its factorization into prime powers. The first factor (or pair of factors, if grouped by parentheses) shows the value that was used in the role of m_1 (as in Section 4.2). The second row gives $\phi(m)$. The third row gives the plaintext space, i.e., the field/ring that is embedded in each slot (here, $R(p^r, d)$ means a ring extension of degree d over \mathbb{Z}_{p^r}). The fourth row gives the number of slots packed into a single ciphertext. The fifth row gives the effective security level, computed using the formula that is used in `HElib`, taken from [15, Eqn.(8)]. The sixth row gives the levels of ciphertext just before recryption (i.e., the ciphertext which is included in the public key) and just after the end of the recryption procedure. The difference accounts for the depth of recryption, and the after-levels is roughly the circuit-depth that an application can compute on the resulting ciphertext before having to recrypt again. We tried to target 10 remaining levels, to allow nontrivial processing between recryptions.

The remaining rows show the resources used in performing a recryption. The timing results reflect the average of the 9 runs for each setting, and the memory usage is the top usage among all these runs. Row 7 gives a one-time initialization cost (all times in seconds). Row 10 (in boldface) gives the total time for a single recryption, while the previous two rows give a breakdown of that time (note that the time for the linear transforms includes some trivial preprocessing time, as well as the less trivial unpacking/repacking time). The last row gives the memory used (in gigabytes).

7 Future work

Eventually, we would like to enhance our implementations to take advantage of multicore computing environments. There are at least two levels at which the implementation could be easily parallelized. At a low level, the conversions between DoubleCRT and polynomial representation in **HElib** could be easily parallelized, as the FFT’s for the different primes can be done in parallel (as already observed in [27]). Our bootstrapping procedure could also be parallelized at a higher level: the rotations in each stage of the linear transformation step can be done in parallel, as can the d different digit extraction steps. Doing the parallel steps at a higher level could possibly yield a better work/overhead ratio, but this would have to be confirmed experimentally.

Another direction to explore is the possibility of speeding up the digit extraction procedure in the special case where the values in the ciphertext slots are constants in the base ring \mathbb{Z}_{p^r} (or, more generally, lie in some sub-ring of the ring E contained in each slot). Right now, our bootstrapping algorithm does not exploit this: even in this special case, our digit extraction algorithm still has to be applied to d ciphertexts. In principle, we should be able to reduce the number of applications of the digit extraction significantly (from d to 1, if the values in the slots are constants); however, it is not clear how to do this while maintaining the structure (and therefore efficiency) of the linear transformations.

Another direction to explore is to try to find a better way to represent constants. In **HElib**, the most compact way to store constants in R_{p^r} is also the most natural: as coefficient vectors of polynomials over \mathbb{Z}_{p^r} . However, in this representation, a surprisingly significant amount of time may be spent in homomorphic computations converting these constants to DoubleCRT format. One could precompute and store these DoubleCRT representations, but this can be quite wasteful of space, as DoubleCRT’s occupy much more space than the corresponding polynomials over \mathbb{Z}_{p^r} . We may state as an open question: is there a more compact representation of elements of $\mathbb{Z}_{p^r}[X]$ that can be converted to DoubleCRT format in linear time?

A Parameters for Digit Extraction

Here we explain our choice of parameters for the recryption procedure (e, e', α , etc.). These parameters depend on the cyclotomic ring R_m , plaintext space p^r , and the l_1 -norm of the recryption secret key $\tilde{\text{sk}}$ (which we denote t).

We begin the recryption procedure with a noise- n ciphertext $(\mathbf{c}_0, \mathbf{c}_1)$, relative to plaintext space p^r , a secret key $\tilde{\text{sk}} = (1, \tilde{\mathbf{s}})$ with $\|\tilde{\mathbf{s}}\|_1 \leq t$, and modulus $\tilde{q} = p^e + 1$. This means that for the element $\mathbf{u} \leftarrow \langle \tilde{\text{sk}}, \mathbf{ct} \rangle$ (without modular reduction) we have $\|[\mathbf{u}]_q\|_\infty < n$ and $\|\mathbf{u}\|_\infty < (t+1)q/2$, and that the plaintext element encrypted in \mathbf{ct} is $m \leftarrow [[\mathbf{u}]_q]_{p^r}$.⁷ We then make the coefficients of \mathbf{ct}

⁷ The term $(t+1)q/2$ assumes no “ring constant”, i.e. $\|\mathbf{s} \cdot \mathbf{c}_1\| \leq \|\mathbf{s}\|_1 \cdot \|\mathbf{c}_1\|$. This is not always true but it makes a reasonable heuristic, and we use it for most of this section.

divisible by $p^{e'}$ using Lemma 3, thus getting another ciphertext

$$\mathbf{ct}' = (\mathbf{c}_0', \mathbf{c}_1') = (\mathbf{c}_0 + p^r \mathbf{u}_0 + q \mathbf{v}_0, \mathbf{c}_1 + p^r \mathbf{u}_1 + q \mathbf{v}_1).$$

Consider the effect of this modification on the coefficients of $\mathbf{u}' \leftarrow \langle \tilde{\mathbf{s}}, \mathbf{ct}' \rangle = \mathbf{c}_0' + \tilde{\mathbf{s}} \cdot \mathbf{c}_1$. Clearly we have increased both the noise (due to the added p^r terms) and the magnitude of the coefficients (mostly due to the added q terms). Specifically, we now have

$$\begin{aligned} \|\mathbf{u}'\| &\leq \|\mathbf{u}\| + (\|\mathbf{u}_0\| + t\|\mathbf{u}_1\|)p^r + (\|\mathbf{v}_0\| + t\|\mathbf{v}_1\|)q \\ &\leq (t+1) \underbrace{\left(\frac{q}{2} + \left\lceil \frac{\alpha p^{r+e'-1}}{2} \right\rceil \right)}_{< q} + qp^r \left(\frac{1}{2} + \left\lfloor \frac{(1-\alpha)p^{e'-1}}{2} \right\rfloor \right) \\ &\leq (t+1)q \left(1 + (1-\alpha)p^{r+e'-1}/2 + p^r/2 \right), \\ \|\mathbf{u}'_q\| &\leq \|\mathbf{u}_q\| + (\|\mathbf{u}_0\| + t\|\mathbf{u}_1\|)p^r \leq n + (t+1)\lceil \alpha p^{r+e'-1}/2 \rceil \\ &\leq n + (t+1)(1 + \alpha p^{r+e'-1}/2). \end{aligned}$$

To be able to still use Lemma 2 we need to have $\|\mathbf{u}'_q\| < q/4$ and $\|\mathbf{u}'\| < q^2/4 - q$. Namely we need both

$$n + (t+1)(1 + \alpha p^{r+e'-1}/2) < q/4 \text{ and } (t+1)(1 + (1-\alpha)p^{r+e'-1}/2 + p^r/2) < q/4 - 1,$$

or in other words

$$q/4 \geq \max \left\{ \frac{(t+1)(1 + \frac{\alpha p^{r+e'-1}}{2})}{(t+1)(1 + \frac{(1-\alpha)p^{r+e'-1}}{2}) + \frac{(t+1)p^r}{2} + 1}, n \right\}. \quad (8)$$

To get good parameters we would like to set α, e' such that these two constraints are roughly equivalent. Ignoring for simplicity the $+1$ at the end of the bottom constraint, we would want to set the parameters so that

$$\begin{aligned} (t+1)\left(1 + \frac{\alpha p^{r+e'-1}}{2}\right) + n &= (t+1)\left(1 + \frac{(1-\alpha)p^{r+e'-1}}{2}\right) + \frac{(t+1)p^r}{2} \\ \Leftrightarrow \alpha &= \frac{1}{2} - \frac{n - (t+1)p^r/2}{(t+1)p^{r+e'-1}}. \end{aligned}$$

Note that with our parameters the noise n is much larger than $(t+1)p^r/2$: The noise after modulus-switching is at least as large as the modulus-switching added factor (cf. [4, Lemma 4]), and the heuristic estimate for that added factor (taken from the **HElib** design document) is $p^r \cdot \sqrt{(t+1)\phi(m)/12}$. Since we use Hamming weight $t \ll \phi(m)$ for the secret key $\tilde{\mathbf{s}}$, then $\sqrt{(t+1)\phi(m)} \gg (t+1)$, which means that $n \approx p^r \cdot \sqrt{(t+1)\phi(m)} \gg p^r \cdot (t+1)$. Hence to get good parameters we need $\alpha \approx \frac{1}{2} - n/((t+1)p^{r+e'-1})$, and since we can only use $\alpha \in [0, 1]$ then it means that we need to set e' large enough in order to get $\alpha > 0$, and α tends to $1/2$ as e' grows.

To get a first estimate, we assume that we have e, e' large enough to get $\alpha \approx 1/2$, and we analyze how large must we make $e - e'$. With $\alpha \approx 1/2$ and the two terms in Eqn. (8) roughly equal, we can simplify that equation to get

$$q/4 = (p^e + 1)/4 > (t + 1)\left(1 + \frac{p^{r+e'-1}}{4} + \frac{p^r}{2}\right) + 1.$$

With $e' \gg 1$ the most significant term on the right-hand side is $(t + 1)p^{r+e'-1}/4$, so we can simplify further to get $p^e/4 > (t + 1 + \epsilon)p^{r+e'-1}/4$ (with ϵ a small quantity that captures all the low-order terms), or $e - e' > r - 1 + \log_p(t + 1 + \epsilon)$. In our implementation we therefore try to use the setting $e - e' = r - 1 + \lceil \log_p(t + 2) \rceil$, and failing that we use $e - e' = r + \lceil \log_p(t + 2) \rceil$.

In more detail, on input m, p, r we set an initial value of $t = 56$, then set $\gamma \stackrel{\text{def}}{=} (t + 1)/p^{\lceil \log_p(t + 2) \rceil}$. Plugging $e - e' = r - 1 + \lceil \log_p(t + 2) \rceil$ in Eqn. (8) and ignoring some ‘+1’ terms, we get

$$p^e > \max \left\{ \frac{4(t + n)}{1 - 2\alpha\gamma}, \frac{2(t + 1)p^r}{1 - 2(1 - \alpha)\gamma} \right\}. \quad (9)$$

For the noise n we substitute twice the modulus-switching added noise term, $n \stackrel{\text{def}}{=} p^r \sqrt{(t + 1)\phi(m)/3}$, and then we solve for the value $\alpha \in [0, 1]$ that minimizes the right-hand side of Eqn. (9). This gives us a lower-bound on p^e .

Next we check that this lower-bound is not too big: recall that at the beginning of the recryption process we multiply the ciphertext \tilde{c} from the public key by the “constant” $c'_1/p^{e'}$, whose entries can be as large as $q^2/(4p^{e'}) \approx p^{2e-e'-2}$. Hence as we increase e we need to multiply by a larger constant, and the noise grows accordingly. In the implementation we define “too big” (somewhat arbitrarily) to be anything more than half the ratio between two successive moduli in our chain. If p^e is “too big” then we reset $e - e'$ to be one larger, which means re-solving the same system but this time using $\gamma' = \gamma/p$ instead of γ .

Once we computed the values e, e', α , we finally check if it is possible to increase our initial $t = 56$ (i.e., the recryption key weight) without violating Eqn. (9). This gives us the final values for all of our constants. We summarize the parameters that we used in our tests in Table 3.

Caveats. The BGV implementation in **HElib** relies on a myriad of parameters, some of which are heuristically chosen, and so it takes some experimentation to set them all so as to get a working implementation with good performance. Some of the adjustments that we made in the course of our testing include the following:

- **HElib** relies on a heuristic noise estimate in order to decide when to perform modulus-switching. One inaccuracy of that estimate is that it assumes that $\|xy\| \leq \|x\| \cdot \|y\|$, which does not quite hold for the bases that are used in **HElib** for representing elements in the ring $R = \mathbb{Z}[X]/(\Phi_m(X))$. To compensate, the library contains a “ring constant” c_m which is set by default to 1 but can be adjusted by the calling application, and then it sets

Table 3. Parameters in our different tests. B is the width (in bits) of levels in the modulus-chain, and c_m is the experimental “ring constant” that we used.

m	p^r	e	e'	α	t	B	c_m	Comments
21854	2	15	9	0.45311	56	23	16.0	
18631	2	15	9	0.45291	56	23	0.5	
28679	2	15	9	0.45241	56	23	10.0	
35115	2	13	6	0	59	25	4.0	“conservative” flag
45551	17	4	2	0	134	25	20.0	
51319	127	3	2	0	56	25	2.0	forced $t = 56$
42799	2^8	23	10	0.45149	57	25	0.2	frequent mod-switching
49981	2^8	23	10	0.45125	57	25	1.0	frequent mod-switching

$\text{estimate}(\|xy\|) := \text{estimate}(\|x\|) \cdot \text{estimate}(\|y\|) \cdot c_m$. In our tests we often had to set that constant to a larger value to get accurate noise estimation — we set the value experimentally so as to get good estimate for the noise at the output of the recryption procedure.

- The same “ring constant” might also affect the setting of the parameters e, e', α from above. Rather than trying to incorporate it into the calculation, our implementation just provides a flag that forces us to forgo the more aggressive setting of $e - e' = r - 1 + \lceil \log_p(t+2) \rceil$, and instead always use the more conservative $e - e' = r + \lceil \log_p(t+2) \rceil$. The effect is that we have to extract one more digit during the digit extraction part, but it ensures that we do not get recryption errors from the use of our simplified decryption formula. In our tests we had to use this “conservative” flag for the tests at $m = 35113$.
- Also, we sometimes had to manually set the Hamming weight of the recryption key to a lower value than what our automatic procedure suggests, to avoid recryption errors. This happened for the setting $p = 127, m = 51319$, where the automated procedure suggested to use $t = 59$ but we had to revert back to $t = 56$ to avoid errors.
- The “width” of each level (i.e., the ratio q_{i+1}/q_i in the modulus chain) can be adjusted in **HElib**. The trade-off is that wider levels give better noise reduction, but also larger overall moduli (and hence lower levels of security). The **HElib** implementation uses by default 23 bits per level, which seems to work well for values of $m < 30000$ and $p^r = 2$. For our tests, however, this was sometime not enough, and we had to increase it to 25 bits per level. For the tests with plaintext space modulo 2^8 , even 25 bits per level were not quite enough. However for various low-level reasons (having to do with the NTL single-precision bounds), setting the bit length to 26 bits or more is not a good option. Instead we changed some of the internals of **HElib**, making it use modulus-switching a little more often than its default setting, while keeping the level width at 25 bits. As a result, for that setting we used many more levels than for all the other settings (an average of 1.5 levels per squaring).

B Why We Didn't Use Ring Switching

One difference between our implementation and the procedure described by Alperin-Sheriff and Peikert [1] is that we do not use the ring-switching techniques of Gentry et al. [13] to implement the tensor decomposition of our `Eval` transformation and its inverse. There are several reasons why we believe that an implementation based on ring switching is less appealing in our context, especially for the smaller parameter settings (say, $\phi(m) < 30000$). The reasoning behind this is as follows:

Rough factorization of m . Since the non-linear part of our recryption procedure takes at least seven levels, and we target having around 10 levels left at the end of recryption, it means that for our smaller examples we cannot afford to spend too many levels for the linear transformations. Since every stage of the linear transformation consumes at least half a level,⁸ then for such small parameters we need very few stages. In other words, we have to consider fairly coarse-grained factorization of m , where the factors have sizes m^ϵ for a significant ϵ (as large as \sqrt{m} in some cases).

Using large rings. Recall that the first linear transformation during recryption begins with the fresh ciphertext in the public key (after multiplying by a constant). That ciphertext has very low noise, so we have to process it in a large ring to ensure security.⁹ This means that we must *switch up* to a much larger ring before we can afford to drop these rough factors of m . Hence we will be spending most of our time on operations in very large rings, which defeats the purpose of targeting these smaller sub-30000 rings in the first place.

We also note that in our tests, the recryption time is dominated by the non-linear part, so our implantation seems close to optimal there. It is plausible that some gains can be made by using ring switching for the second linear transformation, after the non-linear part, but we did not explore this option in our implementation. And as we said above, there is not much to be gained by optimizing the linear transformations.

References

1. Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 1–20. Springer, Heidelberg (2013)
2. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 297–314. Springer, Heidelberg (2014)

⁸ Whether or not we use ring-switching, each stage of the linear transformation has depth of at least one multiply-by-constant, which consumes at least half a level in terms of added noise.

⁹ More specifically, the key-switching matrices that allow us to process it must be defined in a large ring.

3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory **6**(3), 13 (2014)
5. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Naor, M. (ed.) Innovations in Theoretical Computer Science, ITCS 2014, pp. 1–12. ACM (2014)
6. Cheon, J.H., Coron, J.-S., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch fully homomorphic encryption over the integers. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 315–335. Springer, Heidelberg (2013). http://dx.doi.org/10.1007/978-3-642-38348-9_20
7. Coron, J., Lepoint, T., Tibouchi, M.: Batch fully homomorphic encryption over the integers. IACR Cryptology ePrint Archive 2013 **36** (2013). <http://eprint.iacr.org/2013/036>
8. Coron, J.-S., Naccache, D., Tibouchi, M.: Public key compression and modulus switching for fully homomorphic encryption over the integers. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 446–464. Springer, Heidelberg (2012). http://dx.doi.org/10.1007/978-3-642-29011-4_27
9. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-13190-5_2
10. Ducas, L., Micciancio, D.: FHE Bootstrapping in less than a second. Cryptology ePrint Archive, Report 2014/816 (2014). <http://eprint.iacr.org/>
11. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st ACM Symposium on Theory of Computing - STOC 2009. pp. 169–178. ACM (2009)
12. Gentry, C., Halevi, S.: Implementing gentry’s fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011)
13. Gentry, C., Halevi, S., Peikert, C., Smart, N.P.: Field switching in BGV-style homomorphic encryption. Journal of Computer Security **21**(5), 663–684 (2013)
14. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (2012). <http://eprint.iacr.org/2011/566>
15. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012). <http://eprint.iacr.org/2012/099>
16. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 1–16. Springer, Heidelberg (2012)
17. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013)
18. Halevi, S., Shoup, V.: Algorithms in **HElib**. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 554–571. Springer, Heidelberg (2014). <http://eprint.iacr.org/2014/106>

19. Halevi, S., Shoup, V.: HElib - An Implementation of homomorphic encryption. <https://github.com/shaih/HElib/> (Accessed September 2014)
20. 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)
21. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: STOC, pp. 1219–1234 (2012)
22. 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)
23. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM **60**(6), 43 (2013). early version in EUROCRYPT 2010
24. Orsini, E., van de Pol, J., Smart, N.P.: Bootstrapping BGV ciphertexts with a wider choice of p and q. Cryptology ePrint Archive, Report 2014/408 (2014), <http://eprint.iacr.org/>
25. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6) (2009)
26. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation, pp. 169–177. Academic Press (1978)
27. Rohloff, K., Cousins, D.B.: A scalable implementation of fully homomorphic encryption built on NTRU. 2nd Workshop on Applied Homomorphic Cryptography and Encrypted Computing, WAHC 2014 (2014), https://www.dcsec.uni-hannover.de/fileadmin/ful/mitarbeiter/brenner/wahc14_RC.pdf (accessed September 2014)
28. Roman, S.: Field Theory, 2nd edn. Springer (2005)
29. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Cryptography **71**(1), 57–81 (2014). <http://eprint.iacr.org/2011/133>

Efficient Two-Party Protocols

More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries

Gilad Asharov^{1(✉)}, Yehuda Lindell², Thomas Schneider³,
and Michael Zohner³

¹ The Hebrew University of Jerusalem, Jerusalem, Israel
asharov@cs.huji.ac.il

² Bar-Ilan University, Ramat Gan, Israel
lindell@biu.ac.il

³ TU Darmstadt, Darmstadt, Germany
{thomas.schneider,michael.zohner}@ec-spride.de

Abstract. Oblivious transfer (OT) is one of the most fundamental primitives in cryptography and is widely used in protocols for secure two-party and multi-party computation. As secure computation becomes more practical, the need for practical large scale oblivious transfer protocols is becoming more evident. Oblivious transfer extensions are protocols that enable a relatively small number of “base-OTs” to be utilized to compute a very large number of OTs at low cost. In the semi-honest setting, Ishai et al. (CRYPTO 2003) presented an OT extension protocol for which the cost of each OT (beyond the base-OTs) is just a few hash function operations. In the malicious setting, Nielsen et al. (CRYPTO 2012) presented an efficient OT extension protocol for the setting of active adversaries, that is secure in the random oracle model.

In this work, we present an OT extension protocol for the setting of malicious adversaries that is more efficient and uses less communication than previous works. In addition, our protocol can be proven secure in both the random oracle model, and in the standard model with a type of correlation robustness. Given the importance of OT in many secure computation protocols, increasing the efficiency of OT extensions is another important step forward to making secure computation practical.

Keywords: Oblivious transfer extensions · Concrete efficiency · Secure computation

This work was partially supported by the European Union’s Seventh Framework Program (FP7/2007-2013) grant agreement n. 609611 (PRACTICE). The first author is supported by the Israeli Centers of Research Excellence (I-CORE) Program (Center No. 4/11). The second is supported by the European Research Council under the European Union’s Seventh Framework Programme (FP/2007-2013) / ERC consolidators grant agreement n. 615172 (HIPS). The third and fourth authors are supported by the DFG as part of project E3 within the CRC 1119 CROSSING, by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE, and by the Hessian LOEWE excellence initiative within CASED.

1 Introduction

1.1 Background

Oblivious Transfer (OT), introduced by Rabin [33], is a fundamental cryptographic protocol involving two parties, a sender and a receiver. In the most commonly used 1-out-of-2 version [9], the sender has a pair of messages (x_0, x_1) and the receiver has a selection bit r ; at the end of the protocol the receiver learns x_r (but nothing about x_{1-r}) and the sender learns nothing at all about r . Oblivious transfer is a fundamental tool for achieving secure computation, and plays a pivotal role in the Yao protocol [35] where OT is needed for every bit of input of the client, and in the GMW protocol [12] where OT is needed for every AND gate in the Boolean circuit computing the function.

Protocols for secure computation provide security in the presence of adversarial behavior. A number of adversary models have been considered in the literature. The most common adversaries are: *passive* or *semi-honest adversaries* who follow the protocol specification but attempt to learn more than allowed by inspecting the protocol transcript, and *active* or *malicious adversaries* who run any arbitrary strategy in an attempt to break the protocol. In both these cases, the security of a protocol guarantees that nothing is learned by an adversary beyond its legitimate output. Another notion is that of security in the presence of *covert adversaries*; in this case the adversary may follow any arbitrary strategy, but is guaranteed to be caught with good probability if it attempts to cheat. The ultimate goal in designing efficient protocols is to construct protocols that are secure against strong (active or covert) adversaries while adding very little overhead compared to the passive variant. In our paper, we focus primarily on the case of active adversaries, but also provide a variant for covert security.

OT Extensions. As we have mentioned, OT is used extensively in protocols for secure computation. In many cases, this means several millions of oblivious transfers must be run, which can become prohibitively expensive. Specifically, the state-of-the-art protocol for achieving OT with security in the presence of active adversaries of [31] achieves approximately 350 random OTs per second on standard PCs. However, a million OTs at this rate would take over 45 minutes. In order to solve this problem, OT extensions [4] can be used. An OT extension protocol works by running a small number of “*base-OTs*” depending on the security parameter (e.g., a few hundred) that are used as a base for obtaining many OTs via the use of cheap symmetric cryptographic operations only. This is conceptually similar to public-key encryption where instead of encrypting a large message using RSA, which would be too expensive, a hybrid encryption scheme is used such that the RSA computation is only carried out to encrypt a symmetric key, which is then used to encrypt the large message. Such an OT extension can be achieved with extraordinary efficiency; specifically, the protocol of [16] for passive adversaries requires only three hash function computations per OT (beyond the initial base-OTs). In [1], by applying additional algorithmic and cryptographic optimizations, the cost of OT extension for passive adversaries is so low that essentially the communication is the bottleneck. To be concrete,

10,000,000 OTs on random inputs (which suffices for many applications) can be carried out in just 2.62 seconds over a LAN with four threads [1].

For active adversaries, OT extensions are somewhat more expensive. Prior to this work, the best protocol known for OT extensions with security against active adversaries was introduced by [30]. The computational cost of the protocol is due to the number of base-OTs needed for obtaining security, the number of symmetric operations (e.g., hash function computations) needed for every OT in the extension, and the bandwidth. Relative to the passive OT extension of [16], the run-time of [30] is approximately 4 times longer spent on the base-OTs, 1.7 times the cost for each OT in the extension, and 2.7 times the communication. Asymptotically, regarding the number of base-OTs, for security parameter κ (e.g., $\kappa = 128$), it suffices to run κ base-OTs in the passive case. In contrast, [30] require $\lceil \frac{8}{3} \kappa \rceil$ base-OTs.

Applications of OT for Malicious Adversaries. Most prominently, OT is heavily used in today’s most efficient protocols for *secure computation* that allow two or more parties to securely evaluate a function expressed as Boolean circuit on their private inputs. Examples include Yao’s garbled circuits-based approaches such as [10, 11, 14, 19, 22, 24, 25, 32, 34] where OTs are needed for each input, or the Tiny-OT [21, 30] and MiniMac protocols [5, 6] where OTs are needed for each AND gate. Additional applications include the *private set intersection* protocol of [7] which is based purely on OT, and the Yao-based *zero-knowledge* protocol of [17] which allows a party to prove in zero-knowledge a predicate expressed as Boolean circuit, and needs one OT per bit of the witness.

In many of the above applications, the number of oblivious transfers needed can be huge. For instance, for many applications of practical interest, the two-party and multiparty protocols of [5–7, 21, 30] can require several hundred millions of OTs, making the cost of OT the bottleneck in the protocol. Concretely, the current implementations of secure computation in the malicious setting requires $\sim 2^{19}$ OTs for the AES circuit and $\sim 2^{30}$ OTs for the PSI circuit (Sort-Compare-Shuffle), see full version [2] for further details. Thus, improved OT extensions immediately yield faster two-party and multi-party protocols for secure computation.

1.2 Our Contributions

In this paper, we present a new protocol for OT extensions with security in the presence of malicious adversaries, which outperforms the most efficient existing protocol of [30]. We follow the insights of prior work [1, 11], which show that the bottleneck for efficient OT extension is the communication, and focus on decreasing the communication at the cost of slightly increased computation. Furthermore, our protocol can be instantiated with different parameters, allowing us to tradeoff communication for computation. This is of importance since when running over a LAN the computation time is more significant than when running over a WAN where the communication cost dominates. We implement and compare our protocol to the semi-honest protocol of [16] (with optimizations

of [1, 18]) and the malicious protocol of [30] (with optimizations of [11]). As can be seen from the summary of our results given in Table 1, our actively secure protocol performs better than the previously fastest protocol of [30] running at under 60% the cost of the base-OTs of [30], 70% of the cost of each OT in the extension, and 55% of the communication in the local setting. Due to the lower communication, the improvement of our protocol over [30] in the cloud setting (between US East and Europe and thus with higher latency), is even greater with approximately 45% of the time of the base-OTs and 55% of the time for each OT in the extension.

Comparing our protocol to the passive OT extension of [16], our actively secure protocol in the local (LAN) setting costs only 133% more run-time in the base-OTs, 20% more run-time for each OT in the extension, and 50% more communication. In the cloud setting, the cost for each OT in the extension is 63% more than [16] (versus 293% more for [30]). Finally, we obtain covert security at only a slightly higher cost than passive security (just 10% more for each OT in the extension in the local setting, and 30% more in the cloud setting). Our protocol reduces the number of base-OTs that are required to obtain malicious security from $\frac{8}{3}\kappa$ for [30] to $\kappa + \epsilon\rho$, where ρ is the statistical security parameter (e.g., $\rho=40$) and $\epsilon \geq 1$ is a parameter for trading between computation and communication. To be concrete, for $\kappa=128$ -bit security, our protocol reduces the number of base-OTs from 342 to 190 in the local and to 174 in the cloud setting.

Table 1. Run-time and communication for t random OT extensions with $\kappa=128$ -bit security (amortized over 2^{26} executions; [31] amortized over 2^{14} executions). 1KB=8,192bit.

Prot.	Security	Run-Time		Communication	
		Local	Cloud	Local	Cloud
[16]	passive	0.3s+1.07 μ s · t	0.7s+4.24 μ s · t	4KB+128bit · t	
This	covert	0.6s+1.18 μ s · t	1.2s+5.48 μ s · t	21KB+166bit · t	
[20]	active		-	-	42KB+106,018bit · t
[31]	active	2975.32 μ s · t	4597.27 μ s · t	0.3KB+1,024bit · t	
[30]	active	1.2s+1.82 μ s · t	2.9s+12.43 μ s · t	43KB+342bit · t	
This	active	0.7s+1.29 μ s · t	1.3s+6.92 μ s · t	24KB+191bit · t	22KB+175bit · t

In addition to being more efficient, we can prove the security of a variant of our protocol with a version of correlation robustness (where the secret value is chosen with high min-entropy, but not necessarily uniformly), and do not require a random oracle (see §3.3). In contrast, [30] is proven secure in the random oracle model.¹ Our implementation is available online at <http://crypto.de/code/OTExtension> and was integrated into the SCAPI library [8] available at <https://github.com/cryptobiu/scapi>.

¹ It is conjectured that the [30] OT can be proven secure without a random oracle, but this has never been proven.

1.3 Related Work

The first efficient OT extension protocol for semi-honest adversaries was given in [16]. Improvements and optimizations to the protocol of [16] were given in [1, 18].

Due to its importance, a number of previous works have tackled the question of OT extensions with security for malicious/active adversaries. There exist several approaches for achieving security against active adversaries for OT extensions. All of the known constructions build on the semi-honest protocol of [16], and add *consistency checks* of different types to the OT extension protocol, to ensure that the receiver sent consistent values. (Note that in [16], the sender cannot cheat and so it is only necessary to enforce honest behavior for the receiver.)

The first actively-secure version of OT extension used a cut-and-choose technique and was already given in [16]. This cut-and-choose technique achieves a security of 2^{-n} by performing n parallel evaluations of the basic OT extension protocol.

This was improved on by [13, 29], who show that active security can be achieved at a much lower cost. Their approach works in the random oracle model and ensures security against a malicious receiver by adding a low-cost check per extended OT, which uses the uncertainty of the receiver in the choice bit of the sender. As a result, a malicious receiver who wants to learn p choice bits of the sender risks being caught with probability 2^{-p} . However, this measure allows a malicious sender to learn information about the receiver's choice bits. They prevent this attack by combining $S \in \{2, 3, 4\}$ OTs and ensuring the security of one OT by sacrificing the remaining $S - 1$ OTs. Hence, their approach adds an overhead of at least $S \geq 2$ compared to the semi-honest OT extension protocol of [16] for a reasonable number of OTs (with $S = 2$ and approximately 10^7 OTs, they achieve security except with probability 2^{-25} , cf. [29]). However, the exact complexity for this approach has not been analyzed.

An alternative approach for achieving actively-secure OT extension was given in [30]. Their approach also works in the random oracle model but, instead of performing checks per extended OT as in [13, 29], they perform consistency checks per base-OT. Their consistency check method involves hashing the strings that are transferred in the base-OTs and is highly efficient. In their approach, they ensure the security of a base-OT by sacrificing another base-OT, which adds an overhead of factor 2. In addition, a malicious receiver is able to learn p choice bits of the sender with probability 2^{-p} . [30] shows that this leakage can be tolerated by increasing the number of base-OTs from κ to $\lceil \frac{4}{3}\kappa \rceil$. Overall, their approach increases the number of base-OTs that has to be performed by a *multiplicative factor* of $\frac{8}{3}$. The [30] protocol has been optimized and implemented on a GPU in [11]. We give a full description of the [30] protocol with optimizations of [11] in Appendix §B.

An approach for achieving actively-secure OT extension that works in the standard model has recently been introduced in [20]. Their approach achieves less overhead in the base-OTs at the expense of substantially more communication during the check routine (cf. Table 1), and is therefore considerably less efficient.

Nevertheless, we point out that the work of [20] is of independent interest since it is based on the original correlation robustness assumption only.

Since it is the previous best, we compare our protocol to that of [30]. Our approach reduces the number of base-OTs by removing the “sacrifice” step of [30] (where one out of every 2 base-OTs are opened) but increases the workload in the consistency check routine. Indeed, we obtain an additive factor of a statistical security parameter, instead of the multiplicative increase of [30]. This can be seen as a trade-off between reducing communication through fewer base-OTs while increasing computation through more work in the consistency check routine. We empirically show that this results in a more efficient actively secure OT extension protocol, which only has 20% more time and 50% more communication than the passively secure OT extension protocol of [16] in the local setting.

The above works all consider the concrete efficiency of OT extensions. The theoretical feasibility of OT extensions was established in [4], and further theoretical foundations were laid in [26].

2 Preliminaries

2.1 Notation

Our protocol uses a computational (symmetric) security parameter κ and a statistical security parameter ρ . Asymptotically, this means that our protocols are secure for any adversary running in time $\text{poly}(\kappa)$, except with probability $\mu(\kappa) + 2^{-\rho}$. (Formally, the output distribution of a real protocol execution can be distinguished from the output distribution of an ideal execution of the OT functionality with probability at most $\mu(\kappa) + 2^{-\rho}$. See [23] for a formal definition of secure computation with both a statistical and computational security parameter.) In our experiments we set $\kappa = 128$ and $\rho = 40$, which is considered to be secure beyond 2020².

2.2 Oblivious Transfer

Oblivious transfer (OT) was first introduced by Rabin [33] as a function where a receiver receives a message, sent by a sender, with probability 1/2, while the sender remains oblivious whether the message was received. It was later redefined to the functionality more commonly used today by [9], where a sender inputs two messages (x_0, x_1) and the receiver inputs a choice bit r and obliviously receives x_r without learning any information about x_{1-r} . Formally, the 1-out-of-2 OT functionality on n bit strings is defined as $OT_n((x_0, x_1), r) = (\lambda, x_r)$ where λ denotes the empty string and $x_0, x_1 \in \{0, 1\}^n$. In this paper we focus on the general (and most applicable) functionality, which is equivalent to m invocations of the 1-out-of-2 OT functionality on n bit strings. That is, the sender holds as input m pairs of n -bit strings (x_j^0, x_j^1) for $1 \leq j \leq m$ and the receiver holds m

² According to the summary of cryptographic key length recommendations at <http://keylength.com>.

selection bits $\mathbf{r} = (r_1, \dots, r_m)$. The output of the receiver is $(x_1^{r_1}, \dots, x_m^{r_m})$ while the sender has no output. We denote this functionality as $m \times OT_n$. The parties are called sender P_S and receiver P_R .

Several protocols for OT based on different cryptographic assumptions and attacker models were introduced. Most notable are the passive-secure OT protocol of [28] and the active-secure OT protocol of [31], which are among the most efficient today. However, the impossibility result of [15] showed that OT protocols require costly asymmetric cryptography, which greatly limits their efficiency.

2.3 OT Extension

In his seminal work, Beaver [4] introduced *OT extension* protocols, which extend few costly *base-OTs* using symmetric cryptography only. While the first construction of [4] was inefficient and mostly of theoretical interest, the protocol of [16] showed that OT can be extended efficiently and with very little overhead.

Recently, the passively secure OT extension protocol of [16] was improved by [1, 18] who showed how the communication from P_R to P_S can be reduced by a factor of two. Furthermore, [1] implemented and optimized the protocol and demonstrated that the main bottleneck for semi-honest OT extension has shifted from computation to communication. We give the passively secure OT extension protocol of [16] with optimizations from [1, 18] in Protocol 1.

2.4 On the Malicious Security of [16]

The key insight to understanding how to secure OT extension against malicious adversaries is to understand that a malicious party only has very limited possibilities for an attack. In fact, the original OT extension protocol of [16] already provides security against a malicious P_S . In addition, the only attack for a malicious P_R is in Step 2a of Protocol 1, where P_R computes and sends $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$ (cf. [16]). A malicious P_R could choose a different \mathbf{r} for each \mathbf{u}^i (for $1 \leq i \leq \ell$), and thereby extract P_S 's choice bits \mathbf{s} . Hence, malicious security can be obtained if P_R can be forced to use the same choice bits \mathbf{r} in all messages $\mathbf{u}^1, \dots, \mathbf{u}^\ell$.

3 Our Protocol

All we add to the semi-honest protocol (Protocol 1) is a consistency check for the values \mathbf{r} that are sent in Step 2a, and increase the number of base-OTs. Let $\mathbf{r}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{u}^i$, i.e., the value that is implicitly defined by \mathbf{u}^i . We observe that if the receiver P_R uses the same choice bits \mathbf{r}^i and \mathbf{r}^j for some distinct $i, j \in [\ell]^2$, they cancel out when computing their XOR, i.e., $\mathbf{u}^i \oplus \mathbf{u}^j = (\mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}^i) \oplus (\mathbf{t}^j \oplus G(\mathbf{k}_j^1) \oplus \mathbf{r}^j) = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus G(\mathbf{k}_j^0) \oplus G(\mathbf{k}_j^1)$. After the base-OTs, P_S holds $G(\mathbf{k}_i^{s_i})$ and $G(\mathbf{k}_j^{s_j})$ and in Step 2a of Protocol 1, P_R computes and sends $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}^i$ and $\mathbf{u}^j = G(\mathbf{k}_j^0) \oplus G(\mathbf{k}_j^1) \oplus \mathbf{r}^j$. Now note that P_S can compute the XOR of the strings he received in the base-OTs

PROTOCOL 1 (Passive-secure OT extension protocol of [16])

- **Input of P_S :** m pairs (x_j^0, x_j^1) of n -bit strings, $1 \leq j \leq m$.
- **Input of P_R :** m selection bits $\mathbf{r} = (r_1, \dots, r_m)$.
- **Common Input:** Symmetric security parameter κ and $\ell = \kappa$.
- **Oracles and cryptographic primitives:** The parties use an ideal $\ell \times OT_\kappa$ functionality, pseudorandom generator $G : \{0, 1\}^\kappa \rightarrow \{0, 1\}^m$ and a correlation robust-function $H : [m] \times \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ (see §3.3).

1. *Initial OT Phase:*

- (a) P_S initializes a random vector $\mathbf{s} = (s_1, \dots, s_\ell) \in \{0, 1\}^\ell$ and P_R chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ .
- (b) The parties invoke the $\ell \times OT_\kappa$ -functionality, where P_S acts as the *receiver* with input \mathbf{s} and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \leq i \leq \ell$.

For every $1 \leq i \leq \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \dots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its i th column is \mathbf{t}^i for $1 \leq i \leq \ell$. Let \mathbf{t}_j denote the j th row of T for $1 \leq j \leq m$.

2. *OT Extension Phase:*

- (a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends \mathbf{u}^i to P_S for every $1 \leq i \leq \ell$.
- (b) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
- (c) Let $Q = [\mathbf{q}^1 | \dots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its i th column is \mathbf{q}^i . Let \mathbf{q}_j denote the j th row of the matrix Q . (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$)
- (d) P_S sends (y_j^0, y_j^1) for every $1 \leq j \leq m$, where:

$$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$$

- (e) For $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.

- 3. **Output:** P_R outputs $(x_1^{r_1}, \dots, x_m^{r_m})$; P_S has no output.

$G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j})$ as well as the “inverse” XOR of the strings received in the base-OTs $G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}) = G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}) \oplus \mathbf{u}^i \oplus \mathbf{u}^j$ if and only if P_R has correctly used $\mathbf{r}^i = \mathbf{r}^j$. However, P_S cannot check whether the “inverse” XOR is correct, since it has no information about $G(\mathbf{k}_i^{\overline{s_i}})$ and $G(\mathbf{k}_j^{\overline{s_j}})$ (this is due to the security of the base-OTs that guarantees that P_S receives the keys $\mathbf{k}_i^{s_i}, \mathbf{k}_i^{s_j}$ only, and learns nothing about $\mathbf{k}_i^{\overline{s_i}}, \mathbf{k}_j^{\overline{s_j}}$). We solve this problem by having P_R commit to the XORs of all strings $h_{i,j}^{p,q} = H(G(\mathbf{k}_i^p) \oplus G(\mathbf{k}_j^q))$, for all combinations of $p, q \in \{0, 1\}$. Now, given $h_{i,j}^{s_i, s_j}$, $h_{i,j}^{\overline{s_i}, \overline{s_j}}$, P_S checks that $h_{i,j}^{s_i, s_j} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}))$, and that $h_{i,j}^{\overline{s_i}, \overline{s_j}} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}) \oplus \mathbf{u}^i \oplus \mathbf{u}^j) = H(G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}))$. This check passes if $\mathbf{r}^i = \mathbf{r}^j$ and $h_{i,j}^{p,q}$ were set correctly.

If a malicious P_R tries to cheat and has chosen $\mathbf{r}^i \neq \mathbf{r}^j$, it has to convince P_S by computing $h_{i,j}^{p,q} = H(G(\mathbf{k}_i^p) \oplus G(\mathbf{k}_j^q) \oplus \mathbf{r}^i \oplus \mathbf{r}^j)$ for all $p, q \in \{0, 1\}$. However, P_S

can check the validity of $h_{i,j}^{s_i,s_j} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}))$ while P_R remains oblivious to s_i, s_j . Hence, P_R can only convince P_S by guessing s_i, s_j , computing $h_{i,j}^{s_i,s_j}$ correctly and $h_{i,j}^{\bar{s}_i,\bar{s}_j} = H(G(\mathbf{k}_i^{\bar{s}_i}) \oplus G(\mathbf{k}_j^{\bar{s}_j}) \oplus \mathbf{r}^i \oplus \mathbf{r}^j)$, which P_R cannot do better than with probability 1/2. This means that P_R can only successfully learn ρ bits but will be caught except with probability $2^{-\rho}$. The full description of our new protocol is given in Protocol 2. We give some more explanations regarding the possibility of the adversary to cheat during the consistency check in §3.1.

We note that learning few bits of the secret \mathbf{s} does not directly break the security of the protocol once $|\mathbf{s}| > \kappa$. In particular, the values $\{H(\mathbf{t}_j \oplus \mathbf{s})\}_j$ are used to mask the inputs $\{x_j^{1-r_j}\}_j$. Therefore, when H is modelled as a random oracle and enough bits of \mathbf{s} remain hidden from the adversary, each value $H(\mathbf{t}_j \oplus \mathbf{s})$ is random, and the adversary cannot learn the input $x_j^{1-r_j}$. For simplicity we first prove security of our protocol in the random-oracle model. We later show that H can be replaced with a variant of a correlation-robustness assumption.

The advantage of our protocol over [30] is that P_S does not need to reveal any information about s_i, s_j when checking the consistency between r^i and r^j (as long as P_R does not cheat, in which case it risks getting caught). Hence, it can force P_R to check that \mathbf{r}^i equals any \mathbf{r}^j , for $1 \leq j \leq \ell$ without disclosing any information.

Section Outline. In the following, we describe our basic protocol and prove its security (§3.1). We then show how to reduce the number of consistency checks to achieve better performance (§3.2), and how to replace the random oracle with a weaker correlation robustness assumption (§3.3). Finally, we show how our protocol can be used to achieve covert security (§3.4).

3.1 The Security of Our Protocol

Malicious Sender. The original OT extension protocol of [16] already provides security against a malicious P_S . Our checks do not add any capabilities for a malicious sender, since they consist of messages from the receiver to the sender only. Thus, by a simple reduction to the original protocol, one can show that our protocol is secure in the presence of a malicious sender.

Simulating a Malicious Receiver. In the case of a malicious receiver, the adversary may not use the same \mathbf{r} in the messages $\mathbf{u}^1, \dots, \mathbf{u}^\ell$, and as a result learn some bits from the secret \mathbf{s} . Therefore, we add a consistency check of \mathbf{r} to the semi-honest protocol of [16]. However, this verification of consistency of \mathbf{r} is not perfectly sound, and the verification may still pass even when the receiver sends few \mathbf{u} 's that do not define the same \mathbf{r} . This makes the analysis a bit more complicated.

For every $1 \leq i \leq \ell$, let $\mathbf{r}^i \stackrel{\text{def}}{=} \mathbf{u}^i \oplus G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1)$ that is, the “input” \mathbf{r}^i which is implicitly defined by \mathbf{u}^i and the base-OTs.

PROTOCOL 2 (Our active-secure OT extension protocol)

- **Input of P_S :** m pairs (x_j^0, x_j^1) of n -bit strings, $1 \leq j \leq m$.
- **Input of P_R :** m selection bits $\mathbf{r} = (r_1, \dots, r_m)$.
- **Common Input:** Symmetric security parameter κ and statistical security parameter ρ . It is assumed that $\ell = \kappa + \rho$.
- **Oracles and cryptographic primitives:** The parties use an ideal $\ell \times OT_\kappa$ functionality, pseudorandom generator $G : \{0, 1\}^\kappa \rightarrow \{0, 1\}^m$, and random-oracle H (see §3.3 for instantiation of H .)

1. Initial OT Phase:

- (a) P_S initializes a random vector $\mathbf{s} = (s_1, \dots, s_\ell) \in \{0, 1\}^\ell$ and P_R chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ .
- (b) The parties invoke the $\ell \times OT_\kappa$ -functionality, where P_S acts as the *receiver* with input \mathbf{s} and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \leq i \leq \ell$.

For every $1 \leq i \leq \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \dots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its i th column is \mathbf{t}^i for $1 \leq i \leq \ell$. Let \mathbf{t}_j denote the j th row of T for $1 \leq j \leq m$.

2. OT Extension Phase (Part I):

- (a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends \mathbf{u}^i to P_S for every $1 \leq i \leq \ell$.

3. Consistency Check of \mathbf{r} : (the main change from Protocol 1)

- (a) For every pair $\alpha, \beta \subseteq [\ell]^2$, P_R defines the four values:

$$\begin{aligned} h_{\alpha, \beta}^{0,0} &= H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^0)) & h_{\alpha, \beta}^{0,1} &= H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^1)), \\ h_{\alpha, \beta}^{1,0} &= H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^0)) & h_{\alpha, \beta}^{1,1} &= H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^1)). \end{aligned}$$

It then sends $\mathcal{H}_{\alpha, \beta} = (h_{\alpha, \beta}^{0,0}, h_{\alpha, \beta}^{0,1}, h_{\alpha, \beta}^{1,0}, h_{\alpha, \beta}^{1,1})$ to P_S .

- (b) For every pair $\alpha, \beta \subseteq [\ell]^2$, P_S knows $s_\alpha, s_\beta, \mathbf{k}_\alpha^{s_\alpha}, \mathbf{k}_\beta^{s_\beta}, \mathbf{u}^\alpha, \mathbf{u}^\beta$ and checks that:
 - i. $h_{\alpha, \beta}^{s_\alpha, s_\beta} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}))$.
 - ii. $h_{\alpha, \beta}^{\overline{s_\alpha}, \overline{s_\beta}} = H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta) \quad (= H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{r}^\alpha \oplus \mathbf{r}^\beta))$.
 - iii. $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$.

In case one of these checks fails, P_S aborts and outputs \perp .

4. OT Extension Phase (Part II):

- (a) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
- (b) Let $Q = [\mathbf{q}^1 | \dots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its i th column is \mathbf{q}^i . Let \mathbf{q}_j denote the j th row of the matrix Q . (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.)
- (c) P_S sends (y_j^0, y_j^1) for every $1 \leq j \leq m$, where:

$$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$$

- (d) For $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.

- 5. **Output:** P_R outputs $(x_1^{r_1}, \dots, x_m^{r_m})$; P_S has no output.

We now explore how the matrices Q, T are changed when the adversary uses inconsistent \mathbf{r} 's. Recall that when the receiver uses the same \mathbf{r} , then $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$. However, in case of inconsistent \mathbf{r} 's, we get that $\mathbf{q}^i = (s_i \cdot \mathbf{r}^i) \oplus \mathbf{t}^i$. The case of \mathbf{q}_j is rather more involved; let $R = [\mathbf{r}^1 | \dots | \mathbf{r}^\ell]$ denote the $m \times \ell$ matrix where its i th column is \mathbf{r}^i , and let \mathbf{r}_j denote the j th row of the matrix R . For two strings of the same length $\mathbf{a} = (a_1, \dots, a_k), \mathbf{b} = (b_1, \dots, b_k)$, let $\mathbf{a} * \mathbf{b}$ define the entry-wise product, that is $\mathbf{a} * \mathbf{b} = (a_1 \cdot b_1, \dots, a_k \cdot b_k)$. We get that $\mathbf{q}_j = (\mathbf{r}_j * \mathbf{s}) \oplus \mathbf{t}_j$ (note that in an honest execution, \mathbf{r}_j is the same bit everywhere). The sender masks the inputs (x_j^0, x_j^1) with $(H(j, \mathbf{q}_j), H(j, \mathbf{q}_j \oplus \mathbf{s}))$.

In order to understand better the value \mathbf{q}_j , let $\mathbf{r} = (r_1, \dots, r_m)$ be the string that occurs the most from the set $\{\mathbf{r}^1, \dots, \mathbf{r}^\ell\}$, and let $\mathcal{U} \subset [\ell]$ be the set of all indices for which $\mathbf{r}^i = \mathbf{r}$ for all $i \in \mathcal{U}$. Let $B = [\ell] \setminus \mathcal{U}$ be the complementary set, that is, the set of all indices for which for every $i \in B$ it holds that $\mathbf{r}^i \neq \mathbf{r}$. As we will see below, except with some negligible probability, the verification phase guarantees that $|\mathcal{U}| \geq \ell - \rho$. Thus, for every $1 \leq j \leq m$, the vector \mathbf{r}_j (which is the j th row of the matrix R), can be represented as $\mathbf{r}_j = (r_j \cdot \mathbf{1}) \oplus \mathbf{e}_j$, where $\mathbf{1}$ is the all one vector of size ℓ , and \mathbf{e}_j is some error vector with Hamming distance at most ρ from $\mathbf{0}$. Note that the non-zero indices in \mathbf{e}_j are all in B . Thus, we conclude that:

$$\mathbf{q}_j = (\mathbf{s} * \mathbf{r}_j) \oplus \mathbf{t}_j = (\mathbf{s} * (r_j \cdot \mathbf{1} \oplus \mathbf{e}_j)) \oplus \mathbf{t}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j).$$

Recall that in an honest execution $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$, and therefore the only difference is the term $(\mathbf{s} * \mathbf{e}_j)$. Moreover, note that $\mathbf{s} * \mathbf{e}_j$ completely hides all the bits of \mathbf{s} that are in \mathcal{U} , and may expose only the bits that are in B . Thus, the consistency check of \mathbf{r} guarantees two important properties: First, that almost all the inputs are consistent with some implicitly defined string \mathbf{r} , and thus the bits r_j are uniquely defined. Second, the set of inconsistent inputs (i.e., the set B) is small, and thus the adversary may learn only a limited amount of bits of \mathbf{s} .

The Consistency Checks of \mathbf{r} . We now examine what properties are guaranteed by our consistency check, for a single pair (α, β) . The malicious receiver P_R first sends the set of keys $\mathcal{K} = \{\mathbf{k}_i^0, \mathbf{k}_i^1\}$ to the base-OT protocol, and then sends all the values $(\mathbf{u}^1, \dots, \mathbf{u}^\ell)$ and the checks $\mathcal{H} = \{\mathcal{H}_{\alpha, \beta}\}_{\alpha, \beta}$. In the simulation, the simulator can choose \mathbf{s} only after it receives all these messages (this is because the adversary gets no output from the invocation of the OT primitive). Thus, for a given set of messages that the adversary outputs, we can ask what is the number of secrets \mathbf{s} for which the verification will pass, and the number for which it will fail. If the verification passes for some given $\mathcal{T} = (\mathcal{K}, \mathbf{u}^1, \dots, \mathbf{u}^\ell, \mathcal{H})$ and some secret \mathbf{s} , then we say that \mathcal{T} is consistent with \mathbf{s} ; In case the verification fails, we say that \mathcal{T} is inconsistent.

The following Lemma considers the values that the adversary has sent regarding some pair (α, β) , and considers the relation to the pair of bits (s_α, s_β) of the secret \mathbf{s} . We have:

Lemma 31. *Let $\mathcal{T}_{\alpha, \beta} = \{\mathcal{H}_{\alpha, \beta}, \mathbf{u}^\alpha, \mathbf{u}^\beta, \{\mathbf{k}_\alpha^b\}_b, \{\mathbf{k}_\beta^b\}_b\}$ and assume that H is a collision-resistant hash-function. We have:*

1. If $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and $\mathcal{T}_{\alpha,\beta}$ is consistent with (s_α, s_β) , then it is inconsistent with $(\overline{s_\alpha}, \overline{s_\beta})$.
2. If $\mathbf{r}^\alpha = \mathbf{r}^\beta$ and $\mathcal{T}_{\alpha,\beta}$ is consistent with (s_α, s_β) , then it is consistent also with $(\overline{s_\alpha}, \overline{s_\beta})$.

Proof: For the first item, assume that $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and that $\mathcal{T}_{\alpha,\beta}$ is consistent both with (s_α, s_β) and $(\overline{s_\alpha}, \overline{s_\beta})$. Thus, from the check of consistency of (s_α, s_β) :

$$h_{\alpha,\beta}^{s_\alpha, s_\beta} = H \left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \right), \quad h_{\alpha,\beta}^{\overline{s_\alpha}, \overline{s_\beta}} = H \left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta \right),$$

and that $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$. In addition, from the check of consistency of $(\overline{s_\alpha}, \overline{s_\beta})$ it holds that:

$$h_{\alpha,\beta}^{\overline{s_\alpha}, \overline{s_\beta}} = H \left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \right), \quad h_{\alpha,\beta}^{s_\alpha, s_\beta} = H \left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta \right),$$

and that $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$. This implies that:

$$H \left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \right) = h_{\alpha,\beta}^{s_\alpha, s_\beta} = H \left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta \right),$$

and from the collision resistance property of H we get that:

$$G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) = G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta.$$

Recall that $\mathbf{r}^\alpha \stackrel{\text{def}}{=} \mathbf{u}^\alpha \oplus G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1)$, and $\mathbf{r}^\beta \stackrel{\text{def}}{=} \mathbf{u}^\beta \oplus G(\mathbf{k}_\beta^0) \oplus G(\mathbf{k}_\beta^1)$. Combining the above, we get that $\mathbf{r}^\alpha = \mathbf{r}^\beta$, in contradiction. ■

For the second item, once $\mathbf{r}^\alpha = \mathbf{r}^\beta$, we get that $\mathbf{u}^\alpha \oplus \mathbf{u}^\beta = G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^0) \oplus G(\mathbf{k}_\beta^1)$ and it is easy to see that if the consistency check of (s_α, s_β) holds, then the consistency check of $(\overline{s_\alpha}, \overline{s_\beta})$ holds also. ■

Lemma 31 implies what attacks the adversary can do, and what bits of s it can learn from each such an attack. In the following, we consider a given partial transcript $\mathcal{T}_{\alpha,\beta} = ((\mathbf{k}_\alpha^0, \mathbf{k}_\alpha^1, \mathbf{k}_\beta^0, \mathbf{k}_\beta^1), (\mathbf{u}^\alpha, \mathbf{u}^\beta), \mathcal{H}_{\alpha,\beta})$ and analyze what the messages might be, and what the adversary learns in case the verification passes. Let $\mathbf{r}^\alpha = \mathbf{u}^\alpha \oplus G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1)$ and \mathbf{r}^β defined analogously. We consider 4 types:

1. **Type 1: correct.** In this case, it holds that $\mathbf{r}^\alpha = \mathbf{r}^\beta$, and for every $(a, b) \in \{0, 1\}^2$: $h_{\alpha,\beta}^{a,b} = H \left(G(\mathbf{k}_\alpha^a) \oplus G(\mathbf{k}_\beta^b) \right)$. The verification passes for every possible value of (s_α, s_β) .
2. **Type 2: $\mathbf{r}^\alpha = \mathbf{r}^\beta$, but $\mathcal{H}_{\alpha,\beta}$ is incorrect.** In this case, the adversary sent $\mathbf{u}^\alpha, \mathbf{u}^\beta$ that define the same \mathbf{r} . However, it may send hashes $\mathcal{H}_{\alpha,\beta}$ that are incorrect (i.e., for some $(a, b) \in \{0, 1\}^2$, it may send: $h_{\alpha,\beta}^{a,b} \neq H(G(\mathbf{k}_\alpha^a) \oplus G(\mathbf{k}_\beta^b))$). However, from Lemma 31, if $\mathbf{r}^\alpha = \mathbf{r}^\beta$ and $\mathcal{H}_{\alpha,\beta}$ is consistent with (s_α, s_β) then it is also consistent with $(\overline{s_\alpha}, \overline{s_\beta})$. Thus, a possible attack of the adversary, for instance, is to send correct hashes for some bits $(0, 0)$ and $(1, 1)$, but incorrect ones for $(0, 1)$ and $(1, 0)$.

The verification will pass with probability 1/2, exactly if (s_α, s_β) are either $(0, 0)$ or $(1, 1)$, but it will fail in the other two cases (i.e., $(1, 0)$ or $(0, 1)$). We therefore conclude that the adversary may learn the relation $s_\alpha \oplus s_\beta$, and gets caught with probability 1/2.

3. **Type 3: $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and $\mathcal{H}_{\alpha,\beta}$ is incorrect in two positions.** In this case, for instance, the adversary can set the values $h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}$ correctly (i.e., $h_{\alpha,\beta}^{0,0} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^0))$ and $h_{\alpha,\beta}^{0,1} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^1))$) and set values $h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1}$, accordingly, such that the verification will pass for the cases of $(s_\alpha, s_\beta) = (0, 0)$ or $(0, 1)$. That is, it sets:

$$h_{\alpha,\beta}^{1,0} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^1) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta)$$

(and it sets $h_{\alpha,\beta}^{1,1}$ in a similar way). In this case, the adversary succeeds with probability 1/2 and learns that $s_\alpha = 0$ in case the verification passes. Similarly, it can guess the value of s_β and set the values accordingly. For conclusion, the adversary can learn whether its guess was correct, and in which case it learns exactly one of the bits s_α or s_β but does not learn anything about the other bit.

4. **Type 4: $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and $\mathcal{H}_{\alpha,\beta}$ is incorrect in three positions.** In this case, the adversary may guess both bits $(s_\alpha, s_\beta) = (a, b)$ and set $h_{\alpha,\beta}^{a,b}$ correctly, set $h_{\alpha,\beta}^{\bar{a},\bar{b}}$ accordingly (i.e., such that the verification will pass for (a, b)), but will fail for any one of the other cases. In this case, the adversary learns the values (s_α, s_β) entirely, but succeeds with probability of at most 1/4.

Note that whenever $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$, the adversary may pass the verification of the pair (α, β) with probability of at most 1/2. This is because it cannot send consistent hashes for all possible values of (s_α, s_β) , and must, in some sense, “guess” either one of the bits, or both (i.e., Type 3 or Type 4). However, an important point that makes the analysis more difficult is the fact that the two checks are not necessarily independent. That is, in case where $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and $\mathbf{r}^\beta \neq \mathbf{r}^\gamma$, although the probability to pass each one of the verification of (α, β) and (β, γ) separately is at most 1/2, the probability to pass both verifications together is higher than 1/4, and these two checks are not independent. This is because the adversary can guess the bit s_β , and set the hashes as in Type 3 in both checks. The adversary will pass these two checks if it guesses s_β correctly, with probability 1/2.

Theorem 32. *Assuming that H is a random oracle, G is a pseudo-random generator, Protocol 2 with $\ell = \kappa + \rho$ securely computes the $m \times OT_n$ functionality in the $\ell \times OT_\kappa$ -hybrid model in the presence of a static malicious adversary, where κ is the symmetric security parameter and ρ is the statistical security parameter.*

Proof Sketch: The simulator \mathcal{S} invokes the malicious receiver and plays the role of the base-OT trusted party and the honest sender. It receives from the adversary its inputs to the base-OTs, and thus knows the values $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^\ell$. Therefore, it can compute all the values $\mathbf{r}^1, \dots, \mathbf{r}^\ell$ when it receives the messages

$\mathbf{u}^1, \dots, \mathbf{u}^\ell$. It computes the set of indices \mathcal{U} , and extracts \mathbf{r} . It then performs the same checks as an honest sender, in Step 3 of Protocol 2, and aborts the execution if the adversary is caught cheating. Then, it sends the trusted party the value \mathbf{r} that it has extracted, and learns the inputs $x_1^{r_1}, \dots, x_m^{r_m}$. It computes \mathbf{q}_j as instructed in the protocol (recall that these \mathbf{q}_j may contain the additional ‘‘shift’’ $\mathbf{s} * \mathbf{e}_j$) and use some random values for all $\{y_j^{\overline{r}_j}\}_{j=1}^m$. The full description of the simulator is given in the full proof in the full version [2].

Since the values $\{y_j^{\overline{r}_j}\}_{j=1}^m$ are random in the ideal execution, and equal $\{x_j^{\overline{r}_j} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})\}$ in the real execution, a distinguisher may distinguish between the real and ideal execution once it makes a query of the form $(j, \mathbf{q}_j \oplus \mathbf{s})$ to the random oracle. We claim, however, that the probability that the distinguisher will make such a query is bounded by $(t+1)/|S|$, where t is the number of queries it makes to the random oracle, and S is the set of all possible secrets \mathbf{s} that are consistent with the view that it receives. Thus, once we show that $|S| > 2^\kappa$, the probability that it will distinguish between the real and ideal execution is negligible in κ .

However, the above description is too simplified. First, if the adversary performs few attacks of Type 2, it learns information regarding \mathbf{s} from the mere fact that the verification has passed. Moreover, recall that $y_j^{r_j} = x_j^{r_j} \oplus H(j, \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j))$, and that the adversary can control the values \mathbf{t}_j and \mathbf{e}_j . Recall that \mathbf{e}_j is a vector that is all zero in positions that are in \mathcal{U} , and may vary in positions that are in B . This implies that by simple queries to the random oracle, and by choosing the vectors \mathbf{e}_j cleverly, the adversary can totally reveal the bits \mathbf{s}_B quite easily. We therefore have to show that the set B is small, while also showing that the set of consistent secrets is greater than 2^κ (that is, $|S| \geq 2^\kappa$).

Let $\mathcal{T} = \{\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^\ell, \mathbf{u}^1, \dots, \mathbf{u}^\ell, \{H_{\alpha,\beta}\}_{\alpha,\beta}\}$, i.e., the values that the adversary gives during the execution of the protocol. Observe that the simulator chooses the secret \mathbf{s} only after \mathcal{T} is determined (since the adversary receives no output from the execution of the base-OT primitive, we can assume that). We divide all possible \mathcal{T} into two sets, $\mathcal{T}_{\text{good}}$ and \mathcal{T}_{bad} , defined as follows:

$$\begin{aligned}\mathcal{T}_{\text{good}} &= \left\{ \mathcal{T} \mid \Pr_{\mathbf{s}} [\text{consistent}(\mathcal{T}, \mathbf{s}) = 1] > 2^{-\rho} \right\} \text{ and} \\ \mathcal{T}_{\text{bad}} &= \left\{ \mathcal{T} \mid \Pr_{\mathbf{s}} [\text{consistent}(\mathcal{T}, \mathbf{s}) = 1] \leq 2^{-\rho} \right\},\end{aligned}$$

where $\text{consistent}(\mathcal{T}, \mathbf{s})$ is a predicate that gets 1 when the verification passes for the transcript \mathcal{T} and the secret \mathbf{s} , and 0 otherwise. The probability is taken over the choice of \mathbf{s} . For a given \mathcal{T} , let $\mathcal{S}(\mathcal{T})$ be the set of all possible secrets $\mathbf{s} \in \{0, 1\}^\ell$, that are consistent with \mathcal{T} . That is: $\mathcal{S}(\mathcal{T}) = \{\mathbf{s} \in \{0, 1\}^\ell \mid \text{consistent}(\mathcal{T}, \mathbf{s}) = 1\}$. Therefore, it holds that: $\Pr_{\mathbf{s}} [\text{consistent}(\mathcal{T}, \mathbf{s}) = 1] = \frac{|\mathcal{S}(\mathcal{T})|}{2^\ell}$, and thus $|\mathcal{S}(\mathcal{T})| = 2^\ell \cdot \Pr_{\mathbf{s}} [\text{consistent}(\mathcal{T}, \mathbf{s}) = 1]$. As a result, for every $\mathcal{T} \in \mathcal{T}_{\text{good}}$, it holds that $|\mathcal{S}(\mathcal{T})| > 2^\ell \cdot 2^{-\rho} = 2^{\ell-\rho}$. This already guarantees that once the adversary sends transcript \mathcal{T} that will pass the verification with high probability, then the number of possible secrets that are consistent with this transcript is quite large, and therefore it is hard to guess the exact secret \mathbf{s} that was chosen.

We claim that if $|\mathcal{U}| \leq \ell - \rho$ (i.e., $|B| > \rho$), then it must hold that $\mathcal{T} \in \mathcal{T}_{\text{bad}}$ and the adversary gets caught with high probability. Intuitively, this is because we have ρ independent checks, $\{(u, b)\}$, where $u \in \mathcal{U}$ and $b \in B$, which are pairwise disjoint. As we saw, here we do have independency between the checks, and the adversary can pass this verification with probability at most $2^{-\rho}$. Thus, once the adversary outputs transcript \mathcal{T} for which $|B| > \rho$, we have that $\mathcal{T} \in \mathcal{T}_{\text{bad}}$.

We conclude that if the adversary outputs \mathcal{T} for which $\mathcal{T} \in \mathcal{T}_{\text{bad}}$ then it gets caught both in the ideal and the real execution, and the simulation is identical. When $\mathcal{T} \in \mathcal{T}_{\text{good}}$, we get that the number of possible secrets is greater than $2^{\ell-\rho}$, and in addition, $|B| < \rho$. This already gives us a proof for the case where $\ell = \kappa + 2\rho$: Even if we give the distinguisher all the bits \mathbf{s}_B (additonal ρ bits), the set of all possible secrets that are consistent with \mathcal{T} is of size $2^{\ell-2\rho} \geq 2^\kappa$.

In the full proof in the full version [2], we show that $\ell = \kappa + \rho$ base-OTs are sufficient. In particular, we show that for a given transcript $\mathcal{T} \in \mathcal{T}_{\text{good}}$ the bits \mathbf{s}_B are *exactly the same* for all the secrets that are consistent with \mathcal{T} . As a result, the at most ρ bits \mathbf{s}_B that we give to the distinguisher do not give it any new information, and we can set $\ell = \kappa + \rho$. ■

3.2 Reducing the Number of Checks

In Protocol 2, in the consistency check of \mathbf{r} , we check all possible pairs $(\alpha, \beta) \in [\ell]^2$. In order to achieve higher efficiency, we want to reduce the number of checks.

Let $G = (V, E)$ be a graph for which $V = [\ell]$, and an edge (α, β) represents a check between \mathbf{r}^α and \mathbf{r}^β . In Protocol 2 the receiver asks for all possible edges in the graph (all pairs). In order to achieve better performance, we would like to reduce the number of pairs that we check. In particular, the protocol is changed so that in Step 3 of Protocol 2 the sender chooses some set of pairs (edges) $E' \subseteq V^2$, and the receiver must respond with the quadruples $\mathcal{H}_{\alpha, \beta}$ for every $(\alpha, \beta) \in E'$ that it has been asked for. The sender continues with the protocol only if all the checks have passed successfully.

Observe that after sending the values $\mathbf{u}^1, \dots, \mathbf{u}^\ell$, the sets \mathcal{U} and B (which are both subsets of $[\ell]$) are implicitly defined. In case that the set B is too large, we want to catch the adversary cheating with probability of at least $1 - 2^{-\rho}$. In order to achieve this, we should have ρ edges between B and \mathcal{U} that are pairwise non-adjacent. That is, in case the adversary defines B that is “too large”, we want to choose a set of edges E' that contains a matching between B and \mathcal{U} of size of at least ρ .

Note, however, that the sender chooses the edges E' with no knowledge whatsoever regarding the identities of \mathcal{U} and B , and thus it needs to choose a graph such that (with overwhelming probability), for any possible choice of a large B , there will be a ρ -matching between B and \mathcal{U} .

In protocol 3 we modify the consistency check of \mathbf{r} that appears in Step 3 of Protocol 2. The sender chooses for each vertex $\alpha \in [\ell]$ exactly μ out-neighbours uniformly at random. We later show that with high probability the set E' that is chosen contains a ρ -matching between the two sets B and \mathcal{U} , even for a very small value of μ (for instance, $\mu = 3$ or even $\mu = 2$).

PROTOCOL 3 (Modification for Protocol 2, Fewer Checks)

The parties run Protocol 2 with the following modifications:

Step 3 – Consistency Check of \mathbf{r} : (modified)

1. P_S chooses μ functions $\phi_0, \dots, \phi_{\mu-1}$ uniformly at random, where $\phi_i : [\ell] \rightarrow [\ell]$. It sends $\phi_0, \dots, \phi_{\mu-1}$ to the receiver P_R .
2. For every pair $\alpha \in [\ell], i \in [\mu]$, let $(\alpha, \beta) = (\alpha, \phi_i(\alpha))$. P_R defines the four values:

$$\begin{aligned} h_{\alpha, \beta}^{0,0} &= H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^0)) & h_{\alpha, \beta}^{0,1} &= H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^1)) , \\ h_{\alpha, \beta}^{1,0} &= H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^0)) & h_{\alpha, \beta}^{1,1} &= H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^1)) . \end{aligned}$$

It then sends $\mathcal{H}_{\alpha, \beta} = (h_{\alpha, \beta}^{0,0}, h_{\alpha, \beta}^{0,1}, h_{\alpha, \beta}^{1,0}, h_{\alpha, \beta}^{1,1})$ to P_S .

3. P_S checks that it receives $H_{\alpha, \phi_i(\alpha)}$ for every $\alpha \in [\ell]$ and $i \in [\mu]$. Then, for each pair $(\alpha, \beta) = (\alpha, \phi(\alpha))$ it checks that:
 - (a) $h_{\alpha, \beta}^{s_\alpha, s_\beta} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}))$.
 - (b) $h_{\alpha, \beta}^{\overline{s_\alpha}, \overline{s_\beta}} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta) \quad (= H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{r}^\alpha \oplus \mathbf{r}^\beta))$.
 - (c) $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$.

In case one of these checks fails, P_S aborts and outputs \perp .

In our modified protocol, the adversary again first outputs $\mathcal{T} = \{\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^\ell, \mathbf{u}^1, \dots, \mathbf{u}^\ell\}$. Then, the set of checks $\Phi = \{\phi_0, \dots, \phi_{\mu-1}\}$ is chosen, and the adversary responds with $\mathcal{H} = \{\{\mathcal{H}_{\alpha, \phi_i(\alpha)}\}_{\alpha, \phi_i}\}$. We can assume that the actual secret \mathbf{s} is chosen only after \mathcal{T}, Φ and \mathcal{H} are determined. Similarly to the proof of Theorem 32, for a given transcript $(\mathcal{T}, \Phi, \mathcal{H})$ and a secret \mathbf{s} , we define the predicate $\text{consistent}((\mathcal{T}, \Phi, \mathcal{H}), \mathbf{s})$ that gets 1 if and only if the verification is passed for the secret \mathbf{s} (that is, that the sender does not output \perp). For a given \mathcal{T} and set of checks Φ , let $\mathcal{H}_{\mathcal{T}, \Phi}$ be the set of responds that maximizes the probability to pass the verification, that is:

$$\mathcal{H}_{\mathcal{T}, \Phi} \stackrel{\text{def}}{=} \text{argmax}_{\mathcal{H}} \{\Pr[\text{consistent}_{\mathbf{s}}((\mathcal{T}, \Phi, \mathcal{H}), \mathbf{s}) = 1]\} .$$

We separate all possible transcripts (\mathcal{T}, Φ) to two sets $\mathcal{T}_{\text{good}}$ and \mathcal{T}_{bad} such that:

$$\begin{aligned} \mathcal{T}_{\text{good}} &= \{(\mathcal{T}, \Phi) \mid \Pr_{\mathbf{s}}[\text{consistent}((\mathcal{T}, \Phi, \mathcal{H}_{\mathcal{T}, \Phi}), \mathbf{s}) = 1] > 2^{-\rho}\} \text{ and} \\ \mathcal{T}_{\text{bad}} &= \{(\mathcal{T}, \Phi) \mid \Pr_{\mathbf{s}}[\text{consistent}((\mathcal{T}, \Phi, \mathcal{H}_{\mathcal{T}, \Phi}), \mathbf{s}) = 1] \leq 2^{-\rho}\} . \end{aligned}$$

Observe that if a pair $(\mathcal{T}, \Phi) \in \mathcal{T}_{\text{bad}}$, then no matter what set \mathcal{H} the adversary sends, it gets caught with probability of at least $1 - 2^{-\rho}$.

The following claim bounds the size of the set B . It states that if the adversary \mathcal{A} outputs \mathcal{T} that defines $|\mathcal{U}| < \kappa$, then with probability $1 - 2^{-\rho}$ the sender will choose Φ such that $(\mathcal{T}, \Phi) \in \mathcal{T}_{\text{bad}}$.

Claim 33. Let \mathcal{T} be as above, and let \mathcal{U} be the largest set of indices such that for every $\alpha, \beta \in \mathcal{U}$, $\mathbf{r}^\alpha = \mathbf{r}^\beta$. Assume that $|\mathcal{U}| < \kappa$. Then, for appropriate choice of parameters $|B|, \mu$, it holds that:

$$\Pr_{\Phi} [(\mathcal{T}, \Phi) \in \mathcal{T}_{\text{bad}}] \geq 1 - 2^{-\rho}.$$

Proof: The partial transcript \mathcal{T} defines the two sets B and \mathcal{U} . Viewing the base-OTs $[\ell]$ as vertices in a graph, and the pairs of elements that are being checked as edges $E' = \{(\alpha, \phi_i(\alpha)) \mid \alpha \in [\ell], i \in [\mu]\}$, we have a bipartite graph $(B \cup \mathcal{U}, E')$ where each vertex has at least μ out edges. We want to show that with probability $1 - 2^{-\rho}$ (over the choice of Φ), there exists a ρ -matching between \mathcal{U} and B . Once there is a ρ -matching, the adversary passes the verification phase with probability of at most $2^{-\rho}$, and thus the pair (\mathcal{T}, Φ) is in \mathcal{T}_{bad} .

In order to show that in a graph there is a ρ -matching between B and \mathcal{U} , we state the following theorem which is a refinement of Hall's well-known theorem (see [27]). Let $N_{\mathcal{U}}(S)$ denote the set of neighbours in \mathcal{U} , for some set of vertices $S \subseteq B$, that is, $N_{\mathcal{U}}(S) = \{u \in \mathcal{U} \mid \exists v \in S, \text{ s.t. } (u, v) \in E'\}$. We have:

Theorem 34. There exists a matching of size ρ between B and \mathcal{U} if and only if, for any set $S \subseteq B$, $|N_{\mathcal{U}}(S)| \geq |S| - |B| + \rho$.

Note that we need to consider only subsets $S \subseteq B$ for which $|S| \geq |B| - \rho$ (otherwise, the condition holds trivially).

The choice of Φ is equivalent to choosing μ out edges for each vertex uniformly. We will show that for every subset of $S \subseteq B$ with $|S| \geq |B| - \rho$, it holds that $|N_{\mathcal{U}}(S)| \geq |S| - |B| + \rho$.

Let $S \subseteq B$ and $T \subseteq \mathcal{U}$. Let $X_{S,T}$ be an indicator random variable for the event that all the out-edges from S go to $B \cup T$, and all the out-edges of $\mathcal{U} \setminus T$ do not go to S (we use the term “out edges” even though the graph is not directed; our intention is simply the edges connecting these parts). As a result, $|N_{\mathcal{U}}(S)| \leq |T|$. Then, the probability that $X_{S,T}$ equals 1 is the probability that all the $\mu \cdot |S|$ out edges of S go to $B \cup T$ only, and all the $\mu \cdot (|\mathcal{U}| - |T|)$ out edges of $\mathcal{U} \setminus T$ go to $\{\ell\} \setminus S$ only. Since we have independency everywhere, we have:

$$\Pr [X_{S,T} = 1] = \left(\frac{|B| + |T|}{\ell} \right)^{|S| \cdot \mu} \cdot \left(\frac{\ell - |S|}{\ell} \right)^{(|\mathcal{U}| - |T|) \cdot \mu}$$

We are interested in the event $\sum X_{S,T}$ for all $S \subseteq B, T \subseteq \mathcal{U}$ s.t. $|B| - \rho \leq |S| \leq |B|, |T| \leq |S| - |B| + \rho$ (denote this condition by $(*)$), and we want to show that it is greater than 0 with very low probability. We have:

$$\Pr \left[\sum_{S,T, \text{ s.t. } (\star)} X_{S,T} > 0 \right] \leq \sum_{S,T \text{ s.t. } (\star)} \Pr [X_{S,T} = 1] \quad (1)$$

$$\leq \sum_{S,T \text{ s.t. } (\star)} \left(\frac{|B| + |T|}{\ell} \right)^{|S| \cdot \mu} \cdot \left(\frac{\ell - |S|}{\ell} \right)^{(|U| - |T|) \cdot \mu} \quad (2)$$

$$= \sum_{|S|=|B|-\rho}^{|B|} \sum_{|T|=0}^{|S|-|B|+\rho} \binom{|B|}{|S|} \cdot \binom{|U|}{|T|} \cdot \left(\frac{|B| + |T|}{\ell} \right)^{|S| \cdot \mu} \cdot \left(\frac{\ell - |S|}{\ell} \right)^{(|U| - |T|) \cdot \mu}$$

We do not provide an asymptotic analysis for this expression since we loose accuracy by using any upper bound for any one of the terms in it. We next compute this expression for some concrete choice of parameters. We note that the use of the union bound in (2) already reduces the tightness of our analysis, which may cause more redundant checks or base-OTs than actually needed. ■

Concrete Choice of Parameters. Claim 33 states that the bound is achieved for an appropriate choice of parameters. We numerically computed the probability in (1) for a variety of parameters, and obtained that the probability is less than $2^{-\rho}$ with $\rho = 40$, for the following parameters:

κ	128								80			
$ B $	62	49	46	44	43	42	41	53	48	46	42	
μ	2	3	4	5	6	8	15	3	4	5	10	
ℓ	190	177	174	172	171	170	169	133	128	125	122	
#-checks	380	531	696	860	1,026	1,360	2,535	399	512	625	1,220	

In Section 4.2, we run empirical tests to see which parameters perform best in which setting. We recall that in case we check all pairs (i.e., Protocol 2), we have either $\ell = \kappa + \rho = 128 + 40 = 168$ base-OTs with $\binom{\ell}{2} = 14,028$ checks, or $\ell = \kappa + \rho = 80 + 40 = 120$ base-OTs with 7,140 checks.

3.3 Correlation Robustness Instead of a Random Oracle

In this section, we show how a correlation robustness assumption (with respect to a high min-entropy source) suffices for proving the security of our protocol.

Correlation Robust Function. We first recall the standard definition of a correlation robust function from [16], as well as a stronger version of the assumption. Let U_ℓ denote the uniform distribution over strings of length ℓ .

Definition 35 (Correlation Robustness). An efficiently computable function $H : \{0,1\}^\kappa \rightarrow \{0,1\}^n$ is correlation robust if it holds that:

$$\{\mathbf{t}_1, \dots, \mathbf{t}_m, H(\mathbf{t}_1 \oplus \mathbf{s}), \dots, H(\mathbf{t}_m \oplus \mathbf{s})\} \stackrel{c}{=} \{U_{m \cdot \kappa + m \cdot n}\}$$

where $\mathbf{t}_1, \dots, \mathbf{t}_m, \mathbf{s} \in \{0, 1\}^\kappa$ are uniformly and independently distributed. H is strongly correlation robust if for every $\mathbf{t}_1, \dots, \mathbf{t}_m \in \{0, 1\}^\kappa$ it holds that:

$$\{H(\mathbf{t}_1 \oplus \mathbf{s}), \dots, H(\mathbf{t}_m \oplus \mathbf{s})\} \stackrel{\text{c}}{\equiv} \{U_{m \cdot n}\}$$

where $\mathbf{s} \in \{0, 1\}^\kappa$ is uniform.

Another way of looking at this is as a type of *pseudorandom function*. Specifically, define $F_{\mathbf{s}}(\mathbf{t}) = H(\mathbf{t} \oplus \mathbf{s})$. Then, H is correlation robust if and only if F is a weak pseudorandom function, and H is strongly correlation robust if and only if F is a (non-adaptive) pseudorandom function. For proving the security of our protocol, we need to consider the above notions but where \mathbf{s} is chosen from a high min-entropy source. Thus, we consider the case where H is also somewhat an extractor.

Let X be a random variable taking values from $\{0, 1\}^\ell$. The *min-entropy* of \mathcal{X} , denoted $H_\infty(\mathcal{X})$, is: $H_\infty(\mathcal{X}) \stackrel{\text{def}}{=} \min_x \left\{ \log \frac{1}{\Pr[\mathcal{X}=x]} \right\} = -\log(\max_x \{\Pr[\mathcal{X}=x]\})$. If a source \mathcal{X} has a min entropy κ we say that \mathcal{X} is a “ κ -source”. For instance, a κ -source may be κ uniform and independent bits, together with some $\ell - \kappa$ fixed bits (in an arbitrary order), or κ uniform bits with some $\ell - \kappa$ bits that dependent arbitrarily on the first random bits. We are now ready to define min-entropy correlation robustness.

Definition 36 (Min-Entropy Correlation Robustness). An efficiently computable function $H : \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ is κ -min-entropy correlation robust if for all (efficiently samplable) κ -sources \mathcal{X} on $\{0, 1\}^\ell$ it holds that:

$$\{\mathbf{t}_1, \dots, \mathbf{t}_m, H(\mathbf{t}_1 \oplus \mathbf{s}), \dots, H(\mathbf{t}_m \oplus \mathbf{s})\} \stackrel{\text{c}}{\equiv} \{U_{m \cdot \ell + m \cdot n}\}$$

where $\mathbf{t}_1, \dots, \mathbf{t}_m$ are chosen uniformly and independently at random from $\{0, 1\}^\ell$, and $\mathbf{s} \leftarrow \mathcal{X}$. H is κ -min-entropy strongly correlation robust if for all (efficiently samplable) κ -sources \mathcal{X} on $\{0, 1\}^\ell$ and every (distinct) $\mathbf{t}_1, \dots, \mathbf{t}_m \in \{0, 1\}^\ell$ it holds that:

$$\{H(\mathbf{t}_1 \oplus \mathbf{s}), \dots, H(\mathbf{t}_m \oplus \mathbf{s})\} \stackrel{\text{c}}{\equiv} \{U_{m \cdot n}\}$$

where $\mathbf{s} \leftarrow \mathcal{X}$.

In Protocol 2, the values that are used to mask the inputs of the sender are $H(\mathbf{t}_j)$, $H(\mathbf{t}_j \oplus \mathbf{s})$ (or, $H(\mathbf{t}_j \oplus (s * \mathbf{e}_j))$, $H(\mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j) \oplus \mathbf{s})$ in case the adversary uses different \mathbf{r}^i 's). Since the receiver is the one that effectively chooses the \mathbf{t}_j 's values, it may choose values that are not distributed uniformly or even choose them maliciously. As a result, we prove the security of Protocol 2 in its current form using the *strong* κ -min-entropy correlation robustness assumption.

However, it is also possible to modify the protocol and rely only on κ -min-entropy correlation robustness, as follows. In Step 4c (of Protocol 2), in each iteration $1 \leq j \leq m$, the sender chooses a random value $\mathbf{d}_j \in \{0, 1\}^\ell$, and sends the values $(\mathbf{d}_j, y_j^0, y_j^1)$, where:

$$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j \oplus \mathbf{d}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{d}_j \oplus \mathbf{s}) .$$

Then, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j \oplus \mathbf{d}_j)$. Since the \mathbf{d}_j values are chosen last, this ensures that the values used inside H are always uniformly distributed. Thus, κ -min-entropy correlation robustness suffices.

In Step 3 of Protocol 2 we also use the function H ; however, the property that is needed from H for these invocations is collision resistance and not correlation robustness. Therefore, to explicitly emphasize the differences between the two assumptions, we say that the parties use a collision resistant function h in Step 3 of the protocol, and a (variant of) correlation robust function in Step 4c.

Theorem 37

1. Assume that H is strongly κ -min-entropy correlation robust, h is a collision resistant function and G is a pseudo-random generator. Then, Protocol 2 securely computes the $m \times OT_n$ functionality in the $\ell \times OT_\kappa$ -hybrid model in the presence of a static malicious adversary.
2. Assume that H is κ -min-entropy correlation robust, h is a collision resistant function and G is a pseudo-random generator. Then, the above-described modified protocol securely computes the $m \times OT_n$ functionality in the $\ell \times OT_\kappa$ -hybrid model in the presence of a static malicious adversary.

A proof for this theorem appears in the full version [2].

3.4 Achieving Covert Security

In this section, we present a more efficient protocol (with fewer base-OTs and checks) with the property that any deviation from the protocol that can result in a breach of security will be detected with probability at least $1/2$. For details on the definition of covert security, we refer to [3]. Our protocol below is secure under the *strong explicit-cheat formulation* with deterrent factor $\epsilon = \frac{1}{2}$.

As in the malicious case, given the set of keys $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}$, and the messages u^1, \dots, u^ℓ , the sets B and \mathcal{U} are implicitly defined, and we want to catch the adversary if its behavior defines a set B with “high” cardinality. Here, in contrast to the malicious case, we will be content with catching the adversary with probability $1/2$, instead of $1 - 2^{-\rho}$ as in the case of malicious adversaries. As we will show below, our approach for the consistency check of \mathbf{r} enables us to achieve a deterrent factor of $1/2$ at the cost of very few consistency checks. Concretely, it will be enough to use 7 checks of pairs only.

The Protocol. In Step 3 of Protocol 2, the sender chooses t random pairs $\{(\alpha_i, \beta_i)\}_{i=1}^t$ uniformly and independently at random, and sends them to the receiver. The receiver sends $\mathcal{H}_{\alpha_i, \beta_i}$ for each pair (α_i, β_i) that it was asked. Then, the sender performs the same checks as in the previous protocol: It checks that the receiver replied with hashes for all the pairs (α_i, β_i) that it was asked for, and that the hashes that were sent are correct (i.e., as in Step 3b of Protocol 2).

The Analysis. Although at first sight the analysis below ignores attacks of Type 2, these attacks are still taken into consideration. This is because whenever the adversary tries to cheat and learn bits of \mathbf{s} where $\mathbf{r}^\alpha = \mathbf{r}^\beta$, it gets

caught doing so with probability $1/2$, which is exactly the deterrent factor. The analysis therefore focuses on the case that the adversary cheats when $|B|$ is “too large”, and shows that when we have t checks and $|B|$ is large enough, then the probability that the adversary passes the verification is less than $1/2$.

We again consider the graph of checks, and let $V = [\ell]$ and the edges are all possible checks. We divide $[\ell]$ to B and \mathcal{U} , and we show that when using t checks, the probability that the adversary succeeds to pass the verification when B is “large” is less than $1/2$.

There are ℓ^2 edges overall, where $2|B| \cdot |\mathcal{U}|$ are edges between B and \mathcal{U} , and $|B|^2 + |\mathcal{U}|^2$ edges are between B and B , or \mathcal{U} and \mathcal{U} . We say that an edge is “good” if it goes between B and \mathcal{U} . Recall that in such a check, the adversary is caught with probability at least $1/2$.

For the first edge that is chosen, the probability that it is a good edge is $2|B| \cdot |\mathcal{U}|/\ell^2$. However, once this specific edge between B and \mathcal{U} is chosen, an edge between B and \mathcal{U} that is pairwise non-adjacent with the previously chosen edge is no longer good, since the probability that the adversary will get caught here is not $1/2$. Therefore, we denote by good_i the probability of choosing the $(i+1)$ th “good” edge. That is, the probability that edge e_j is good, conditioned on the event that i good edges were previously chosen in the set $\{e_1, \dots, e_{j-1}\}$. We have that:

$$\text{good}_i = \frac{2 \cdot (|B| - i) \cdot (|\mathcal{U}| - i)}{\ell^2}.$$

This holds because once a good edge is chosen, we do not want to choose an edge that is adjacent to it. As a result, with each good edge that is chosen, the effective size of the set B and \mathcal{U} is decreased by 1.

In contrast, we denote by bad_i the probability that the next chosen edge is bad, given that there were i previous good edges. That is, a bad edge is either an edge between B and B , an edge between \mathcal{U} and \mathcal{U} , or is adjacent to one of the $2i$ vertices of the previously chosen good edges. This probability is as follows:

$$\text{bad}_i = \frac{|B|^2 + |\mathcal{U}|^2 + 2i \cdot |\mathcal{U}| + 2i \cdot |B| - 2i^2}{\ell^2} = \frac{|B|^2 + |\mathcal{U}|^2 + 2i(\ell - i)}{\ell^2}$$

That is, a bad edge can be either an edge from B to B , \mathcal{U} to \mathcal{U} , or an edge between the i vertices that were chosen with any other vertex. Note, however, that there are some edges that are counted twice and thus we remove $2i^2$. In addition, observe that $\text{good}_i + \text{bad}_i = 1$.

When we have t checks, we may have between 0 to t good edges. In case there are d good edges, the probability that the adversary succeeds to cheat is 2^{-d} . In order to ease the calculation, let good be the maximal probability of $\text{good}_0, \dots, \text{good}_{t-1}$, and let bad be the maximal probability of $\text{bad}_0, \dots, \text{bad}_t$. We get that:

$$\text{good} = \frac{2 \cdot |B| \cdot |\mathcal{U}|}{\ell^2}$$

and for $t < \ell/2$:

$$\text{bad} = \frac{|B|^2 + |\mathcal{U}|^2 + 2t(\ell - t)}{\ell^2}.$$

Now, consider the edges e_1, \dots, e_t . The probability that the adversary succeeds in its cheating is the union of succeeds in cheating in each possible combination of checks. In particular, we may have $d = 0, \dots, t$ good edges, and for each d , there are $\binom{t}{d}$ possible ways to order d good edges and $t - d$ “bad” edges. Finally, when we have d good edges, the probability that the adversary succeeds to cheat is 2^{-d} . We therefore have that the probability that the adversary successfully cheats without being caught is less than:

$$\sum_{d=0}^t \binom{t}{d} \cdot \text{good}^d \cdot \text{bad}^{t-d} \cdot 2^{-d} = \sum_{d=0}^t \binom{t}{d} \cdot \left(\frac{1}{2} \cdot \text{good}\right)^d \cdot \text{bad}^{t-d} = \left(\frac{1}{2} \cdot \text{good} + \text{bad}\right)^t.$$

It is easy to verify that this probability is less than 0.5 for $|B| = 38$, $|\mathcal{U}| = 128$ (and so overall $\ell = 166$), with only 7 checks. In which case, we have that $\text{good} = 0.353$, $\text{bad} = 0.728$, and the probability is less than 0.495.

4 Performance Evaluation

We experimentally compare the performance of our protocols to previous works using the same programming language and running benchmarks on the same machines: We first describe our implementation (§4.1), empirically evaluate and compare the identified active and covert parameters of §3.2 and §3.4 (§4.2), and compare our work to the active-secure protocol of [30] with optimizations of [11] and to the passive-secure protocol of [16] with optimizations from [1] (§4.3).

Benchmarking Environment: We run our experiments in two settings: a local setting and a cloud setting. In the *local setting*, the sender and receiver routines run on two Desktop PCs which each have 16 GB RAM, an Intel Haswell i7-4770K CPU with 4 cores and AES-NI support, and are connected via Gigabit Ethernet. In the *cloud setting*, we run the OT sender routine on an Amazon EC2 m3.medium instance with a 2.5 GHz, Intel Xeon E5-2670v2 CPU and 3.75 memory located in North Virginia (US East) and run the OT receiver routine on one of our Desktop PCs in Europe. The average bandwidth usage in the cloud setting was 52 MBit/s and the average ping latency (round-trip-time) was 95 ms.

4.1 Implementation

We build on the passive-secure and publicly available OT extension C++ implementation of [1]. We perform the OT extension protocol and consistency checks block-wise, i.e., we split m OTs into b blocks of size $w = 2^{18}$, with $b = \lceil \frac{m}{w} \rceil$. These blocks can be processed independently of each other and using multiple threads. For all experiments we evaluate the random OT version of [1], since the additional overhead to obtain the traditional OT functionality is equal for all protocols, and output $n = 8$ -bit strings. For the base-OTs we use [28] for the passive-secure OT extension protocol and [31] in decryption mode with security based on the Decisional Diffie-Hellmann (DDH) assumption for the covert- and active-secure OT extension protocols; we implement both using elliptic curves. We assume $\kappa = 128$ -bit long-term security with $\rho = 40$ statistical security. Further implementation details are given in Appendix §A.

4.2 Parameter Evaluation

We evaluate the asymptotic communication and run-time in the local and cloud setting on 2^{23} random OTs for our most promising active security (cf. Table 3.2) and covert security (cf. §3.4) parameters, and compare them to the active-secure protocol of [30] with $\ell = \lceil \frac{8}{3}\kappa \rceil = 342$ base-OTs and $\ell/2 = 171$ checks, and to the passive-secure protocol of [16] with $\ell = 128$ base-OTs and no checks. The results are depicted in Table 2 where the parameters are given as (#base-OTs;#checks). We also include the pairwise comparison Protocol 2 (which performs all possible checks) with parameters (168;14,028) and discuss its special features in Appendix §A.3.

Table 2. Run-time and communication for active, covert, and passive security using different parameters (#base-OTs;#checks) on 2^{23} random OTs. Minimum values are marked in bold.

Security Parameters	Active						Covert	Passive
[30]	190;380	177;531	174;696	170;1,360	168;14,028		166;7	[16]
Comm. [MB]	342	191	178	175	173	195	166	128
<i>Local Setting</i>								
Run-time [s]	16.988	11.938	13.201	18.218	25.918	221.382	10.675	9.579
<i>Cloud Setting</i>								
Run-time [s]	110.223	64.698	63.845	63.712	83.414	454.595	46.718	33.838

For the communication we can observe that our parameter sets have 50% – 55% of the communication of [30]. Furthermore, while decreasing the number of base-OTs reduces the overall communication until 170 base-OTs, the overhead in communication for sending the consistency check hashes outweighs the gains from the reduced number of base-OTs. Hence, using less than 170 base-OTs for block-size $w = 2^{18}$ would increase both communication and computation complexity.

For the run-time we can observe that our best-performing parameter has 70% of the run-time of [30] in the local setting and 58% of the run-time in the cloud setting. Furthermore, the best-performing parameter differs between the local and cloud setting: while the (190;380) parameter performs best in the local setting, the (174;696) parameter achieves the lowest run-time in the cloud setting. This can be explained by the smaller bandwidth of the cloud setting, which influences the run-time of all parameters differently. For instance, when switching from the local to the cloud setting, the run-time of [30] increases by factor 6.5, whereas that of our pairwise comparison Protocol 2 with parameter (168;14,028) only increases by factor 2. As expected, the covert parameter (166;7) performs better than the parameters for active security.

4.3 Comparison with Related Work

We empirically evaluate and compare our protocol on a varying number of OTs in its active and covert versions to the passive-secure OT extension protocol of [16] with optimizations of [1, 18], and the active-secure OT extension protocol of [30]

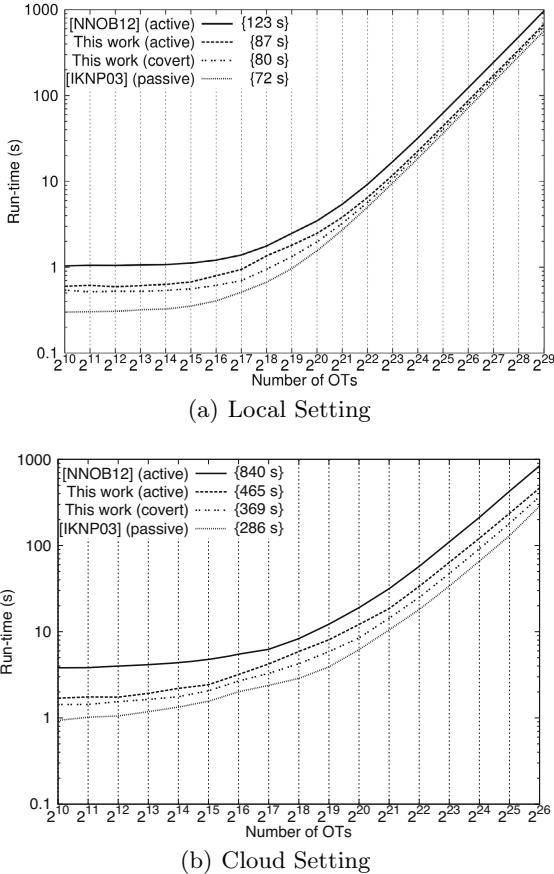


Fig. 1. Run-time for random OT extension protocols for 8-bit strings with active, covert, and passive security in the local- and cloud setting. Time for 2^{26} OTs given in {}.

with optimizations of [11]. The results for the local and cloud setting are given in Figure 1. We benchmark the protocols on an exponentially increasing number of OTs: from 2^{10} to 2^{29} for the local setting and from 2^{10} to 2^{26} for the cloud setting. The passive-secure [16] serves as bottom-line for the performance of the other protocols to show the (small) gap to the covert- and active-secure protocols. For our protocol we use the parameters from our parameter evaluation in §4.2 which were shown to perform best in the respective setting, i.e., (190;380) for the local setting, (174;696) for the cloud setting, and (166;7) for covert security. For the [30] protocol we use $\ell = \lceil \frac{8}{3}\kappa \rceil = 342$ base-OTs and $\ell/2 = 171$ checks. We excluded the active-secure protocol of [20], since its communication overhead is at least two orders of magnitude higher than for the evaluated protocols and

simply transferring the required data would result in higher run-times than those of the other protocols.

For the results in the local setting we can observe that our active-secure OT extension protocol outperforms the [30] protocol for all OTs tested on and scales better with increasing number of OTs. Furthermore, our active-secure protocol converges towards the passive-secure [16] protocol when more OTs are performed, decreasing the overhead for active security down to 121% for 2^{26} OTs, compared to an overhead of 171% for the [30] protocol. The convergence of our protocol can be explained by the amortizing costs of the consistency checks. Since the consistency checks are performed on blocks of fixed width 2^{18} , their amortization happens for a larger number of OTs. The covert version of our protocol has only 111% overhead compared to the passive-secure protocol.

In the cloud setting, the performance of all protocols decreases, as expected. However, the performance of the passive-secure protocol decreases less significantly compared to the covert- and active-secure protocols. This can be explained by the smaller communication complexity of the passive-secure protocol, since the run-time overhead scales with the communication overhead of the respective protocol. For the active-secure protocol of [30] with communication overhead of 267% compared to the passive-secure protocol, the run-time overhead increases from 171% to 294%. In comparison, for our active-secure protocol with communication overhead of 136%, the run-time overhead increases from 121% to 163%. Finally, for our covert protocol with communication overhead of 129%, the run-time overhead increases from 111% to 129%.

References

1. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: ACM Computer and Communications Security (CCS 2013), pp. 535–548. ACM (2013). Code: <http://crypto.de/code/OTExtension>
2. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer extensions with security for malicious adversaries (full version). IACR Cryptology ePrint Archive 2015, 061 (2015). Online: <http://eprint.iacr.org/2015/061>
3. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. Journal of Cryptology **23**(2), 281–343 (2010)
4. Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: Symposium on the Theory of Computing (STOC 1996), pp. 479–488. ACM (1996)
5. Damgård, I., Lauritsen, R., Toft, T.: An empirical study and some improvements of the MiniMac protocol for secure computation. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 398–415. Springer, Heidelberg (2014)
6. Damgård, I., Zakarias, S.: Constant-overhead secure computation of Boolean circuits using preprocessing. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 621–641. Springer, Heidelberg (2013)
7. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: An efficient and scalable protocol. In: ACM Computer and Communications Security (CCS 2013), pp. 789–800. ACM (2013)

8. Eijenberg, Y., Farbstein, M., Levy, M., Lindell, Y.: SCAPI: the secure computation application programming interface. IACR Cryptology ePrint Archive 2012, 629 (2012). Online: <http://eprint.iacr.org/2012/629>
9. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Communications of the ACM **28**(6), 637–647 (1985)
10. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B.: Faster maliciously secure two-party computation using the GPU. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 358–379. Springer, Heidelberg (2014)
11. Frederiksen, T.K., Nielsen, J.B.: Fast and maliciously secure two-party computation using the GPU. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 339–356. Springer, Heidelberg (2013)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Symposium on Theory of Computing (STOC 1987), pp. 218–229. ACM (1987)
13. Harnik, D., Ishai, Y., Kushilevitz, E., Nielsen, J.B.: OT-combiners via secure computation. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 393–411. Springer, Heidelberg (2008)
14. Huang, Y., Katz, J., Kolesnikov, V., Kumaresan, R., Malozemoff, A.J.: Amortizing garbled circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 458–475. Springer, Heidelberg (2014)
15. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 8–26. Springer, Heidelberg (1990)
16. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
17. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently. In: ACM Computer and Communications Security (CCS 2013), pp. 955–966. ACM (2013)
18. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 54–70. Springer, Heidelberg (2013)
19. Kreuter, B., Shelat, A., Shen, C.: Billion-gate secure computation with malicious adversaries. In: USENIX Security Symposium 2012, pp. 285–300. USENIX (2012)
20. Larraia, E.: Extending oblivious transfer efficiently, or - how to get active security with constant cryptographic overhead. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014. LNCS, vol. 8895, pp. 336–384. Springer, Heidelberg (2015). Online: <http://eprint.iacr.org/2014/692>
21. Larraia, E., Orsini, E., Smart, N.P.: Dishonest majority multi-party computation for binary circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 495–512. Springer, Heidelberg (2014)
22. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
23. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer, Heidelberg (2011)
24. Lindell, Y., Pinkas, B., Smart, N.P.: Implementing two-party computation efficiently with security against malicious adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 2–20. Springer, Heidelberg (2008)

25. Lindell, Y., Riva, B.: Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 476–494. Springer, Heidelberg (2014)
26. Lindell, Y., Zarosim, H.: On the feasibility of extending oblivious transfer. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 519–538. Springer, Heidelberg (2013)
27. Lovász, L., Plummer, M.: Matching Theory. Akadémiai Kiadó, Budapest (1986), also published as, Vol. 121 of the North-Holland Mathematics Studies, North-Holland Publishing, Amsterdam
28. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Symposium on Discrete Algorithms (SODA 2001), pp. 448–457. ACM/SIAM (2001)
29. Nielsen, J.B.: Extending oblivious transfers efficiently - how to get robustness almost for free. IACR Cryptology ePrint Archive 2007, 215 (2007). Online: <http://eprint.iacr.org/2007/215>
30. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012)
31. 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)
32. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
33. Rabin, M.O.: How to exchange secrets with oblivious transfer, TR-81 edn. Aiken Computation Lab, Harvard University (1981)
34. Shelat, A., Shen, C.H.: Fast two-party secure computation with minimal assumptions. In: ACM Computer and Communications Security (CCS 2013), pp. 523–534. ACM (2013)
35. Yao, A.C.: How to generate and exchange secrets. In: Foundations of Computer Science (FOCS 1986), pp. 162–167. IEEE (1986)

A Implementation Details

In this section we provide details about the architecture of our implementation (§A.1), the method we use to allow block-wise evaluation of our protocol and [30] (§A.2), and discuss the benefits of the pairwise-comparison method described in Protocol 2 (§A.3).

A.1 Architecture

We designed the architecture of the active-secure OT extension implementations such that the communication-intensive passive-secure OT extension routine and the computation-intensive checks on receiver side are performed by separate threads and can be further parallelized independently of each other. This architecture allows us to instantiate the implementation specifically to the

available resources of the deployment scenario. More detailed, we can perform the communication-intensive operations with as many threads as required to fully utilize the bandwidth and can then focus the remaining processing power on the computationally-intensive operations. This kind of parallelization offers benefits especially for deployment scenarios of OT extension with small bandwidth, where the network is the bottle-neck for OT extension and where further parallelization of communication-intensive operations would only result in congestion on the network interface. Although this architecture favors our protocol which is computationally more intensive than the protocols of [16] and [30], we argue that it nicely fits to today’s increasing number of CPU cores.

A.2 3-Step OT Extension

Note that in order to allow block-wise evaluation of our protocol and [30], the base-OTs have to be renewed. For the block-wise evaluation of $m \times OT_n$ in b blocks of width w bits ($b = \lceil \frac{m}{w} \rceil$), we perform a 3-step OT extension: In the first step, we perform $\ell \times OT_{b\ell}$ base-OTs using the protocol of [31]. In the second step, we extend $\ell \times OT_{b\ell}$ to $bl \times OT_w$ using the respective active secure OT extension protocol. In the third step, we again perform the OT extension step b -times on each ℓ -bit interval, i.e., we extend $\ell \times OT_w$ to $w \times OT_n$ b -times and thereby obtain $bw \geq m$ OTs on n -bit strings.

A.3 Advantages of the Pairwise Comparison Protocol

Although the pairwise comparison Protocol 2 with parameter (168;14,028) is the slowest in our evaluation in §4.2, we stress that it has several advantages which make it favorable in settings with high computation power. The main advantage is that the receiver can pre-compute all checks directly after the base-OTs, since all combinations are checked and hence the sender does not need to send a mapping to the receiver. Additionally, if a computationally powerful device such as a GPU is present, the receiver can use it for computing the checks in parallel.

B Active Secure OT Extension of [30]

In Protocol 4 we depict the actively-secure OT extension protocol of [30] with optimizations from [11].

PROTOCOL 4 (Active secure OT extension protocol of [30])

- **Input of P_S :** m pairs (x_j^0, x_j^1) of n -bit strings, $1 \leq j \leq m$.
- **Input of P_R :** m selection bits $\mathbf{r} = (r_1, \dots, r_m)$.
- **Common Input:** Symmetric security parameter κ and $\ell = \lceil \frac{8}{3}\kappa \rceil$.
- **Oracles and primitives:** Ideal $\ell \times OT_\kappa$ functionality, pseudorandom generator G , correlation-robust function H , and random-oracle H' .

1. *Initial OT Phase:*

- (a) P_S initializes a random vector $\mathbf{s} = (s_1, \dots, s_\ell) \in \{0, 1\}^\ell$ and P_R chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ .
- (b) The parties invoke the $\ell \times OT_\kappa$ -functionality, where P_S acts as the *receiver* with input \mathbf{s} and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \leq i \leq \ell$.

For every $1 \leq i \leq \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \dots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its i th column is \mathbf{t}^i for $1 \leq i \leq \ell$. Let \mathbf{t}_j denote the j th row of T for $1 \leq j \leq m$.

2. *OT Extension Phase:*

- (a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends \mathbf{u}^i to P_S for every $1 \leq i \leq \ell$.
- (b) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.

3. *Consistency Check of \mathbf{r} :*

- (a) P_S chooses a uniform random permutation $\pi : \{1, \dots, \ell\} \mapsto \{1, \dots, \ell\}$ with $\pi(\pi(i)) = i$ and sends π to Bob. Let $\Pi(\pi) = \{i | i \leq \pi(i)\}$.
- (b) For all $i \in \Pi(\pi)$, P_S computes $d_i = s_i \oplus s_{\pi(i)}$ and $\mathbf{z}^i = \mathbf{q}^i \oplus \mathbf{q}^{\pi(i)}$ sends d_i to P_R .
- (c) P_R computes $\mathbf{z}'^i = (d_i \cdot \mathbf{r}) \oplus \mathbf{t}^i \oplus \mathbf{t}^{\pi(i)}$.
- (d) P_S and P_R check equality between $\mathbf{Z} = \mathbf{z}_1 || \dots || \mathbf{z}_{\lfloor \ell/2 \rfloor}$ and $\mathbf{Z}' = \mathbf{z}'_1 || \dots || \mathbf{z}'_{\lfloor \ell/2 \rfloor}$ as follows:
 - i. P_S samples $w \in_R \{0, 1\}^\kappa$, computes $\mathbf{c} = H'(\mathbf{Z} || \mathbf{w})$, sends \mathbf{c} to P_R .
 - ii. P_R then sends \mathbf{Z}' to P_S .
 - iii. P_S checks $\mathbf{Z} \stackrel{?}{=} \mathbf{Z}'$ and aborts on failure. Else sends (\mathbf{Z}, \mathbf{w}) to P_R .
 - iv. P_R checks that $\mathbf{Z} \stackrel{?}{=} \mathbf{Z}'$ and $c \stackrel{?}{=} H'(\mathbf{Z}' || \mathbf{w})$ and aborts on failure.
- (e) For all $\lfloor \ell/2 \rfloor$ indices in $i \in \Pi(\pi)$ where i is the k th index with $1 \leq k \leq \lfloor \ell/2 \rfloor$, P_S sets $\mathbf{q}'_k = \mathbf{q}_i$ and $s'_k = s_i$ and P_R sets $\mathbf{t}'_k = \mathbf{t}_i$.

4. *OT Extension (continue):*

- (a) Let $Q' = [\mathbf{q}'^1 | \dots | \mathbf{q}'^{\lfloor \ell/2 \rfloor}]$ denote the $m \times \lfloor \ell/2 \rfloor$ bit matrix where its i th column is \mathbf{q}'^i . Let \mathbf{q}'_j denote the j th row of the matrix Q' . (Note that $\mathbf{q}'^i = (s'_i \cdot \mathbf{r}) \oplus \mathbf{t}'^i$ and $\mathbf{q}'_j = (r_j \cdot \mathbf{s}') \oplus \mathbf{t}'_j$.)
- (b) P_S sends (y_j^0, y_j^1) for every $1 \leq j \leq m$, where $y_j^0 = x_j^0 \oplus H(j, \mathbf{q}'_j)$ and $y_j^1 = x_j^1 \oplus H(j, \mathbf{q}'_j \oplus \mathbf{s}')$.
- (c) For $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}'_j)$.

5. **Output:** P_R outputs $(x_1^{r_1}, \dots, x_m^{r_m})$; P_S has no output.

How to Efficiently Evaluate RAM Programs with Malicious Security

Arash Afshar^{1(✉)}, Zhangxiang Hu², Payman Mohassel³, and Mike Rosulek²

¹ University of Calgary, Calgary, Canada
aafshar@ucalgary.ca

² Oregon State University, Corvallis, USA
{huz,rosulekm}@eecs.oregonstate.edu

³ Yahoo Labs, Sunnyvale, USA
p.mohassel@yahoo-inc.com

Abstract. Secure 2-party computation (2PC) is becoming practical for some applications. However, most approaches are limited by the fact that the desired functionality must be represented as a boolean circuit. In response, random-access machines (RAM programs) have recently been investigated as a promising alternative representation.

In this work, we present the first practical protocols for evaluating RAM programs with security against malicious adversaries. A useful efficiency measure is to divide the cost of malicious-secure evaluation of f by the cost of semi-honest-secure evaluation of f . Our RAM protocols achieve ratios matching the state of the art for circuit-based 2PC. For statistical security 2^{-s} , our protocol without preprocessing achieves a ratio of s ; our online-offline protocol has a pre-processing phase and achieves online ratio $\sim 2s/\log T$, where T is the total execution time of the RAM program.

To summarize, our solutions show that the “extra overhead” of obtaining malicious security for RAM programs (beyond what is needed for circuits) is minimal and does not grow with the running time of the program.

1 Introduction

General secure two-party computation (2PC) allows two parties to perform “arbitrary” computation on their joint inputs without revealing any information about their private inputs beyond what is deducible from the output of computation. This is an extremely powerful paradigm that allows for applications to utilize sensitive data without jeopardizing its privacy.

From a feasibility perspective, we know that it is possible to securely compute any function, thanks to seminal results of [11, 41]. The last decade has also witnessed significant progress in design and implementation of more practical/scalable secure computation techniques, improving performance by orders of magnitude and enabling computation of circuits with billions of gates.

Mike Rosulek — Supported by NSF award CCF-1149647.

These techniques, however, are largely restricted to functions represented as Boolean or arithmetic circuits, whereas the majority of applications we encounter in practice are more efficiently captured using random-access memory (RAM) programs that allow constant-time memory lookup. Modern algorithms of practical interest (e.g., binary search, Dijkstra’s shortest-paths algorithm, and the Gale-Shapely stable matching algorithm) all rely on fast memory access for efficiency, and suffer from major blowup in running time otherwise. More generally, a circuit computing a RAM program with running time T requires $\Theta(T^2)$ gates in the worst case, making it prohibitively expensive (as a general approach) to compile RAM programs into a circuit and then apply known circuit 2PC techniques.

A promising alternative approach uses the building block of *oblivious RAM*, introduced by Goldreich and Ostrovsky [12]. ORAM is an approach for making a RAM program’s memory access pattern input-oblivious while still retaining fast (polylogarithmic) memory access time. Recent work in 2PC has begun to investigate direct computation of ORAM computations as an alternative to RAM-to-circuit compilation [10, 13, 19, 27, 28]. These works all follow the same general approach of evaluating a sequence of ORAM instructions using traditional circuit-based 2PC phases. More precisely, they use existing circuit-based MPC to (1) initialize and setup the ORAM, a one-time computation with cost proportional to the memory size, (2) evaluate the next-instruction circuit which outputs “shares” of the RAM program’s internal state, the next memory operations (read/write), the location to access, and the data value in case of a write. All of these existing solutions provide security only against semi-honest adversaries.

Challenges for malicious-secure RAM evaluation. It is possible to take a semi-honest secure protocol for RAM evaluation (e.g., [13]) and adapt it to the malicious setting using standard techniques. Doing so naïvely, however, would result in several major inefficiencies that are avoidable. We point out three significant challenges for efficient, malicious-secure RAM evaluation:

1: Integrity and consistency of state information, by which we mean both the RAM’s small internal state and its large memory both of which are passed from one CPU step to the next. A natural approach for handling internal state is to have parties hold secret shares of the state (as in [13]), which they provide as input to a secure evaluation of the next-instruction circuit. Using standard techniques for malicious-secure SFE, it would incur significant overhead in the form of oblivious transfers and consistency checks to deal with state information as inputs to the circuit.

A natural approach suitable for handling RAM memory is to evaluate an Oblivious RAM that encrypts its memory contents. In this approach, the parties must evaluate a next-instruction circuit that includes both encryption and decryption sub-circuits. Evaluating a block cipher’s circuit securely against malicious adversaries is already rather expensive [22], and this approach essentially asks the parties to do so at every time-step, even when the original RAM’s behavior is non-cryptographic. Additional techniques are needed to detect any

tampering of data by either participant, such as computing/verifying a MAC of each memory location access inside the circuit or computing a “shared” Merkle-tree on top of the memory in order to check its consistency after each access. All these solutions incur major overhead when state is passed or memory is accessed and are hence prohibitively expensive (see full version [1] for a concrete example).

2: Compatibility with batch execution and input-recovery techniques. In a secure computation, every input bit must be “touched” at some point. Oblivious RAM programs address this with a pre-processing phase that “touches” the entire (large) RAM memory, after which the computation need not “touch” every bit of memory. Since an offline phase is already inevitable for ORAMs, we would like to use such a phase to further increase the efficiency of the online phase of the secure evaluation protocol. In particular, recent techniques of [15, 26] suggest that pre-processing/batching garbled circuits can lead to significant efficiency improvement for secure evaluation of circuits. The fact that the ORAM next-instruction circuits are used at every timestep and are known *a priori* makes the use of batch execution techniques even more critical.

Another recent technique, called input-recovery [23], reduces the number of garbled circuits in cut-and-choose by a factor of 3 by only requiring that at least one of the evaluated circuits is correct (as opposed to the majority). This is achieved by running an input-recovery step at the end of computation that recover’s the garbler’s private input in case he cheats in more than one evaluated circuit. The evaluator then uses the private input to do the computation on his own. A natural applications of this technique in case of RAM programs, would require running the input-recovering step after every timestep which would be highly inefficient (see full version [1] for a concrete example).

3: Run-time dependence. The above issues are common to any computation that involves persistent, secret internal state across several rounds of inputs/outputs (any so-called *reactive* functionality). RAM programs present an additional challenge, in that only part of memory is accessed at each step, and furthermore these memory locations are determined *only at run-time*. In particular, it is non-trivial to reconcile run-time data dependence with offline batching optimizations.

Our approach: In a RAM computation, both the memory and internal state need to be *secret* and *resist tampering* by a malicious adversary. As mentioned above, the obvious solutions to these problem all incur major overhead whenever state is passed from one execution to the next or memory is accessed. We bypass all these overheads and obtain secrecy and tamper-resistance essentially for free. Our insight is that these are properties also shared by wire labels in most garbling schemes — they hide the associated logical value, and, given only one wire label, it is hard to “guess” the corresponding complementary label.

Hence, instead of secret-sharing the internal state of the RAM program between the parties, we simply “re-use” the garbled wire labels from the output of one circuit into the input of the next circuit. These wire labels already inherit the required authenticity properties, so no oblivious transfers or consistency checks are needed.

Similarly, we also encode the RAM’s memory via wire labels. When the RAM reads from memory location ℓ , we simply reuse the appropriate output wire labels from the most recent circuit to write to location ℓ (not necessarily the previous instruction, as is the case for the internal state). Since the wire labels already hide the underlying logical values, we only require an oblivious RAM that hides the memory access pattern and *not* the contents of memory. More concretely, this means that we do not need to add encryption/decryption and MAC/verify circuitry inside the circuit that is being garbled or perform oblivious transfers on shared intermediate secrets. Importantly, if the RAM program being evaluated is “non-cryptographic” (i.e., has a small circuit description) then the circuits garbled at each round of our protocols will be small.

Of course, it is a delicate task to make these intuitive ideas work with the state of art techniques for cut-and-choose. We present two protocols, which use different approaches for reusing wire labels.

The first protocol uses ideas from the LEGO paradigm [9, 33] for 2PC and other recent works on batch-preprocessing of garbled circuits [15, 26]. The idea behind these techniques is to generate all the necessary garbled circuits in an offline phase (before inputs are selected), open and check a random subset, and randomly assign the rest into buckets, where each bucket corresponds to one execution of the circuit. But unlike the setting of [15, 26], where circuits are processed for many *independent* evaluations of a function, we have the additional requirement that the wire labels for memory and state data should be directly reused between various garbled circuits. Since we cannot know which circuits must have shared wire labels (due to random assignment to buckets and runtime memory access pattern), we use the “soldering” technique of [9, 33] that directly transfers garbled wire labels from one wire to another, after the circuits have been generated. However, we must adapt the soldering approach to make it amenable to soldering entire circuits as opposed to soldering simple gates as in [9, 33]. For a discussion of subtle problems that arise from a direct application of their soldering technique, see Section 3.

Our second approach directly reuses wire labels without soldering. As a result, garbled circuits cannot be generated offline, but the scheme does not require the homomorphic commitments required for the LEGO soldering technique. At a high level, we must avoid having the cut-and-choose phase reveal secret wire labels that are shared in common with other garbled circuits. The technique recently proposed in [31] allows us to use a single cut-and-choose for all steps of the RAM computation (rather than independent cut-and-choose steps for each time step), and further hide the set of opened/evaluated circuits from the garbler using an OT-based cut-and-choose [18, 22]. We observe that this approach is compatible with the state of the art techniques for input-consistency check [30, 38].

We also show how to incorporate the input-recovery technique of [23] for reducing the number of circuits by a factor of three. The naive solution of running the cheating recovery after each timestep would be prohibitively expensive since it would require running a malicious 2PC for the cheating recovery circuit

(and the corresponding input-consistency checks) at every timestep. We show a modified approach that only requires a final cheating recovery step at the end of the computation.

Based on some concrete measurements (see full version [1]), the “extra overhead” of achieving malicious security for RAM programs (i.e. the additional cost beyond what is needed for malicious security of the circuits involved in the computation), is at least an order of magnitude smaller than the naive solutions and this gap grows as the running time of the RAM program increases.

Related work. Starting with seminal work of [11, 42], the bulk of secure multiparty computation protocols focus on functions represented as circuits (arithmetic or Boolean). More relevant to this work, there is over a decade’s worth of active research on design and implementation of *practical* 2PC protocols with malicious security based on garbled circuits [14, 20, 23–25, 29, 30, 37, 38], based on GMW [32], and based on arithmetic circuits [8].

The work on secure computation of RAM programs is much more recent. [13] introduces the idea of using ORAM inside a Yao-based secure two-party computation in order to accommodate (amortized) sublinear-time secure computation. The work of [10, 28] study non-interactive garbling schemes for RAM programs which can be used to design protocols for secure RAM program computation. The recent work of [19], implements ORAM-based computation using arithmetic secure computation protocol of [8], hence extending these ideas to the multiparty case, and implementing various oblivious data-structures. SCVM [27] and Obliv-C [44] provide frameworks (including programming languages) for secure computation of RAM programs that can be instantiated using different secure computation RAM programs on the back-end. The above work all focus on the semi-honest adversarial model. To the best of our knowledge, our work provides the first practical solution for secure computation of RAM program with malicious security. Our constructions can be used to instantiate the back-end in SCVM and Obliv-C with malicious security.

2 Preliminaries

2.1 (Oblivious) RAM Programs

A RAM program is characterized by a deterministic circuit Π and is executed in the presence of memory M . The memory is an array of *blocks*, which are initially set to 0^n . An execution of the RAM program Π on inputs (x_1, x_2) with memory M is given by:

```

 $\text{RAMEval}(\Pi, M, x_1, x_2)$ 
  st :=  $x_1 \| x_2 \| 0^n$ ; block :=  $0^n$ ; inst :=  $\perp$ 
  do until inst has the form (HALT, z):
    block := [if inst = (READ,  $\ell$ ) then  $M[\ell]$  else  $0^n$ ]
     $r \leftarrow \{0, 1\}^n$ ; (st, inst, block) :=  $\Pi(st, block, r)$ 
    if inst = (WRITE,  $\ell$ ) then  $M[\ell] := block$ 
  output z

```

Oblivious RAM, introduced in [12], is a technique for hiding all information about a RAM program’s memory (both its contents and the data-dependent access pattern). Our constructions require a RAM program that hides only the memory access pattern, and we will use other techniques to hide the *contents* of memory. Throughout this work, when we use the term “ORAM”, we will be referring to this weaker security notion. Concretely, such an ORAM can often be obtained by taking a standard ORAM construction (e.g., [7, 40]) and removing the steps where it encrypts/decrypts memory contents.

Define $\mathcal{I}(\Pi, M, x_1, x_2)$ as the random variable denoting the sequence of values taken by the `inst` variable in $\text{RamEval}(\Pi, M, x_1, x_2)$. Our precise notion of ORAM security for Π requires that there exist a simulator \mathcal{S} such that, for all x_1, x_2 and initially empty M , the output $\mathcal{S}(1^\lambda, z)$ is indistinguishable from $\mathcal{I}(\Pi, M, x_1, x_2)$, where z is the final output of the RAM program on inputs x_1, x_2 .

2.2 Garbling Schemes

In this section we adapt the abstraction of *garbling schemes* [5] to our needs. Our 2PC protocol constructions re-use wire labels between different garbled circuits, so we define a specialized syntax for garbling schemes in which the input and output wire labels are pre-specified.

We represent a set of wire labels W as a $m \times 3$ array. Wire labels $W[i, 0]$ and $W[i, 1]$ denote the two wire labels associated with some wire i . We employ the point-permute optimization [34], so we require $\text{lsb}(W[i, b]) = b$. The value $W[i, 2]$ is a single-bit *translation bit*, so that $W[i, W[i, 2]]$ is the wire label that encodes FALSE for wire i . For shorthand, we use $\tau(W)$ to denote the m -bit string $W[1, 2] \dots W[m, 2]$.

We require the garbling scheme to have syntax $F \leftarrow \text{Garble}(f, E, D)$ where f is a circuit, E and D represent wire labels as above.

For $v \in \{0, 1\}^m$, we define $W|_v = (W[1, v_1], \dots, W[m, v_m])$, i.e., the wire labels with *select bits* v . We also define $W|_x^* := W|_{x \oplus \tau(W)}$, i.e., the wire labels corresponding to *truth values* x . The correctness condition we require for garbling is that, for all f, x , and valid wire label descriptions E, D , we have:

$$\text{Eval}(\text{Garble}(F, E, D), E|_x^*) = D|_{f(x)}^*$$

If Y denotes a vector of output wire labels, then it can be decoded to a plain output via $\text{lsb}(Y) \oplus \tau(D)$, where lsb is applied component-wise. Hence, $\tau(D)$ can be used as output-decoding information. More generally, if $\mu \in \{0, 1\}^m$ is a mask value, then revealing $(\mu, \tau(D) \wedge \mu)$ allows the evaluator to learn only the output bits for which $\mu_i = 1$.

Let \mathcal{W} denote the uniform distribution of $m \times 3$ matrices of the above form (wire labels with the constraint on least-significant bits described above). Then the security condition we need is that there exists an efficient simulator \mathcal{S} such that for all f, x, D , the following distributions are indistinguishable:

$$\begin{array}{ll} \text{Real}(f, x, D): & \text{Sim}^S(f, x, D): \\ \hline E \leftarrow \mathcal{W} & E \leftarrow \mathcal{W} \\ F \leftarrow \text{Garble}(f, E, D) & F \leftarrow \mathcal{S}(f, E|_x^*, D|_{f(x)}^*) \\ \text{return } (F, E|_x^*) & \text{return } (F, E|_x^*) \end{array}$$

To understand this definition, consider an evaluator who receives garbled circuit F and wire labels $E|_x^*$ which encode its input x . The security definition ensures that the evaluator learns no more than the correct output wires $D|_{f(x)}^*$.

Consider what happens when we apply this definition with D chosen from \mathcal{W} and against an adversary who is given only partial decoding information $(\mu, \tau(D) \wedge \mu)$.¹ Such an adversary's view is then independent of $f(x) \wedge \bar{\mu}$. This gives us a combination of the *privacy* and *obliviousness* properties of [5]. Furthermore, the adversary's view is independent of the complementary wire labels $D|_{\overline{f(x)}}^*$, except possibly in their least significant bits (by the point-permute constraint). So the other wire labels are hard to predict, and we achieve an *authenticity* property similar to that of [5].²

Finally, we require that it be possible to efficiently determine whether F is in the range of $\text{Garble}(f, E, D)$, given (f, E, D) . For efficiency improvements, one may also reveal a seed which was used to generate the randomness used in Garble .

These security definitions can be easily achieved using typical garbling schemes used in practice (e.g., [21]). We note that the above arguments hold even when the distribution \mathcal{W} is slightly different. For instance, when using the Free-XOR optimization [21], wire label matrices E and D are chosen from a distribution parameterized by a secret Δ , where $E[i, 0] \oplus E[i, 1] = \Delta$ for all i . This distribution satisfies all the properties of \mathcal{W} that were used above.

Conventions for wire labels. We exclusively garble the ORAM circuit which has its inputs/outputs partitioned into several logical values. When W is a description of input wire labels for such a circuit, we let $\text{st}(W)$, $\text{rand}(W)$, $\text{block}(W)$ denote the submatrices of W corresponding to the incoming internal state, random tape, and incoming memory block. When W describes output wires, we use $\text{st}(W)$, $\text{inst}(W)$ and $\text{block}(W)$ to denote the outgoing internal state, output instruction (read/write/halt, and memory location), and outgoing memory data block. We use these functions analogously for vectors (not matrices) of wire labels.

2.3 (XOR-Homomorphic) Commitment

In addition to a standard commitment functionality \mathcal{F}_{com} , one of our protocols requires an XOR-homomorphic commitment functionality $\mathcal{F}_{\text{xcom}}$. This functionality allows P_1 to open the XOR of two or more committed messages without

¹ Our definition applies to this case, since a distinguisher for the above two distributions is allowed to know D which parameterizes the distributions.

² We stress that the evaluator *can indeed decode* the garbled output (using $\tau(D)$ and the select bits), yet *cannot forge* valid output wire labels in their entirety. This combination of requirements was not considered in the definitions of [5].

The functionality is initialized with internal value $i = 1$. It then repeatedly responds to commands as follows:

- On input $(commit, m)$ from P_1 , store (i, m) internally, set $i := i + 1$ and output $(committed, i)$ to both parties.
- On input $(open, S)$ from P_1 , where S is a set of integers, for each $i \in S$ find (i, m_i) in memory. If for some i , no such m_i exists, send \perp to P_2 . Otherwise, send $(open, S, \bigoplus_{i \in S} m_i)$ to P_2 .

Fig. 1. XOR-homomorphic commitment functionality $\mathcal{F}_{\text{xcom}}$

leaking any other information about the individual messages. The functionality is defined in Figure 1. Further details, including an implementation, can be found in [9].

3 Batching Protocol

3.1 High-Level Overview

Roughly speaking, the LEGO technique of [9, 33] is to generate a large quantity of garbled gates, perform a cut-and-choose on all gates to ensure their correctness, and finally assemble the gates together into a circuit which can tolerate a bounded number of faulty gates (since the cut-and-choose will not guarantee that all the gates are correct). More concretely, with sN gates and a cut-and-choose phase which opens half of them correctly, a statistical argument shows that permuting the remaining gates into **buckets** of size $O(s/\log N)$ each ensures that each bucket contains a majority of correct gates, except with negligible probability in s .

For each gate, the garbler provides a *homomorphic commitment* to its input/output wire labels, which is also checked in the cut and choose phase. This allows wires to be connected on the fly with a technique called **soldering**. A wire with labels (w_0, w_1) (here 0 and 1 refer to the public select bits) can be soldered to a wire with labels (w'_0, w'_1) as follows. If w_0 and w'_0 both encode the same truth value, then decommit to $\Delta_0 = w_0 \oplus w'_0$ and $\Delta_1 = w_1 \oplus w'_1$. Otherwise decommit to $\Delta_0 = w_0 \oplus w'_1$ and $\Delta_1 = w_1 \oplus w'_0$. Then when an evaluator obtains the wire label w_b on the first wire, $w_b \oplus \Delta_b$ will be the correct wire label for the second wire. To prove that the garbler hasn't inverted the truth value of the wires by choosing the wrong case above, she must also decommit to the XOR of each wire's *translation* bit (i.e., $\beta \oplus \beta'$ where w_β and $w'_{\beta'}$ both encode false).

Next, an arbitrary gate within each bucket is chosen as the **head**. For each other gate, we solder its input wires to those of the head, and output wires to those of the head. Then an evaluator can transfer the input wire labels to each of

the gates (by XORing with the appropriate solder value), evaluate the gates, and transfer the wire labels back. The majority value is taken to be the output wire label of the bucket. The cut-and-choose ensures that each bucket functions as a correct gate, with overwhelming probability. Then the circuit can be constructed by appropriately soldering together the buckets in a similar way.

For our protocol we use a similar approach but work with buckets of *circuits*, not buckets of gates. Each bucket evaluates a single timestep of the RAM program. To transfer RAM memory and internal state between timesteps, we solder wires together appropriately (i.e., state input of time t soldered to state output of time $t - 1$; memory-block input t soldered to memory-block output of the previous timestep that wrote to the desired location). Additionally, the approach of using buckets also saves an asymptotic $\log T$ factor in the number of circuits needed for each timestep (i.e., the size of the buckets), where T is the total running time of the ORAM, a savings that motivates similar work on batch pre-processing of garbled circuits [15, 26].

We remark that our presentation of the LEGO approach above is a slight departure from the original papers [9, 33]. In those works, all gates were garbled using Free XOR optimization, where $w_0 \oplus w_1$ is a secret constant shared on all wires. Hence, we have only one “solder” value $w_0 \oplus w'_0 = w_1 \oplus w'_1$. If the sender commits to only the “false” wire label of each wire, then the sender is prevented from inverting the truth value while soldering (“false” is always mapped to “false”). However, to keep the offset $w_0 \oplus w_1$ secret, only one of the 4 possible input combinations of each gate can be opened in the cut-and-choose phase. The receiver has only a 1/4 probability of identifying a faulty gate. This approach does not scale to a cut-and-choose of entire circuits, where the number of possible input combinations is exponential. Hence our approach of forgoing common wire offsets $w_0 \oplus w_1$ between circuits and instead committing to the translation bits. As a beneficial side effect, the concrete parameters for bucket sizes are improved since the receiver will detect faulty circuits with probability 1, not 1/4.

Back to our protocol, P_1 generates $O(sT/\log T)$ garблings of the ORAM’s next-instruction circuit, and commits to the circuits and their wire labels. P_2 chooses a random half of these to be opened and aborts if any are found to be incorrect.

For each timestep t , P_2 picks a random subset of remaining garbled circuits and the parties assemble them into a bucket \mathcal{B}_t (this is the `MkBucket` subprotocol) by having P_1 open appropriate XORs of wire labels, as described above. We can extend the garbled-circuit evaluation function `Eval` to `EvalBucket` using the same syntax. Then `EvalBucket` inherits the correctness property of `Eval` with overwhelming probability, for each of the buckets created in the protocol.

After a bucket is created, P_2 needs to obtain garbled inputs on which to evaluate it. See Figure 3 for an overview. Let X_t denote the vector of input wire labels to bucket \mathcal{B}_t . We use $\text{block}(X_t)$, $\text{st}(X_t)$, $\text{rand}(X_t)$ to denote the sets of wire labels for the input memory block, internal state, and shares of random tape, respectively. The simplest wire labels to handle are the ones for internal state,

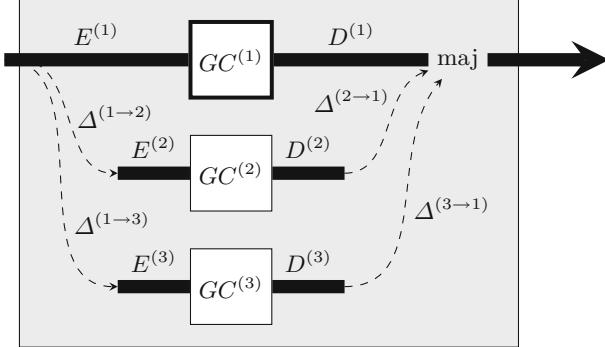


Fig. 2. Illustration of $\text{MkBucket}(\mathcal{B} = \{1, 2, 3\}, \text{hd} = 1)$

as they always come from the previous timestep. We solder the output internal state wires of bucket \mathcal{B}_{t-1} to the input internal state wires of bucket \mathcal{B}_t . Then if Y_{t-1} were the output wire labels for bucket \mathcal{B}_{t-1} by P_2 , we obtain $\text{st}(X_t)$ by adjusting $\text{st}(Y_{t-1})$ according to the solder values.

If the previous memory instruction was a READ of a location that was last written to at time t' , then we need to solder the appropriate output wires from bucket $\mathcal{B}_{t'}$ to the corresponding input wires of \mathcal{B}_t . P_2 then obtains $\text{block}(X_t)$ by adjusting the wire labels $\text{block}(Y_{t'})$ according to the solder values. If the previous memory instruction was a READ of an uninitialized block, or a WRITE, then P_1 simply opens these input wire labels to all zero values (see $\text{GetInput}_{\text{pub}}$).

To obtain wire labels $\text{rand}(X_t)$, we have P_1 open wire labels for its shares (GetInput_1) and have P_2 obtain its wire labels via a standard OT (GetInput_2).

At this point, P_2 can evaluate the bucket (EvalBucket). Let Y_t denote the output wire labels. P_1 opens the commitment to their translation values, so P_2 can decode and learn these outputs of the circuit. P_2 sends these labels back to P_1 , who verifies them for authenticity. Knowing only the translation values and not the entire actual output wire labels, P_2 cannot lie about the circuit's output except with negligible probability.

3.2 Detailed Protocol Description

Let Π be the ORAM program to be computed. Define $\tilde{\Pi}(\text{st}, \text{block}, \text{inp}_1, \text{inp}_{2,1}, \dots, \text{inp}_{2,n}) = \Pi(\text{st}, \text{block}, \text{inp}_1, \bigoplus_i \text{inp}_{2,i})$. Looking ahead, during the first timestep, the parties will provide $\text{inp}_1 = x_1$ and $\text{inp}_2 = x_2$, while in subsequent timesteps they input their shares r_1 and r_2 of the RAM program's randomness. P_2 's input is further secret shared to prevent a selective failure attack on both x_2 and his random input r_2 . We first define the following subroutines / subprotocols:

prot Solder(A, A') // A, A' are wire labels descriptions

P_1 opens $\mathcal{F}_{\text{xcom}}$ -commitments to $\tau(A)$ and $\tau(A')$

so that P_2 receives $\tau = \tau(A) \oplus \tau(A')$

for each position i in τ and each $b \in \{0, 1\}$:

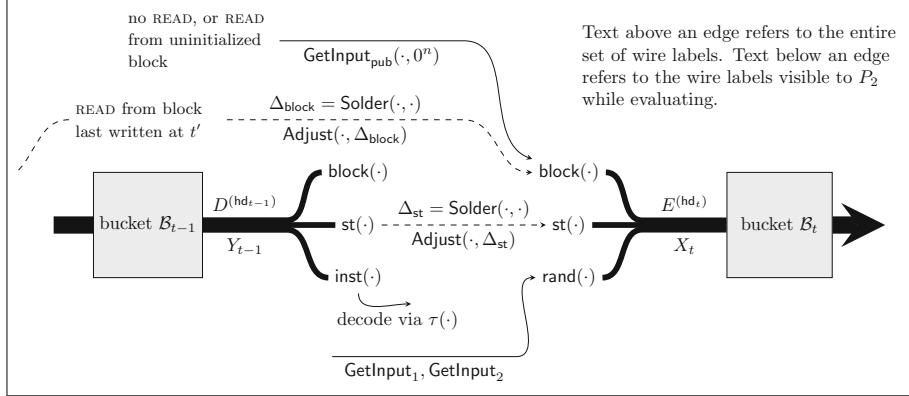


Fig. 3. Overview of soldering and evaluation steps performed in the online phase

P_1 opens $\mathcal{F}_{\text{xcom}}$ -commitments to $A[i, b]$ and $A'[i, \tau_i \oplus b]$

so that P_2 receives $\Delta[i, b] = A[i, b] \oplus A'[i, \tau_i \oplus b]$

return Δ

prot MkBucket(\mathcal{B} , hd) // \mathcal{B} is a set of indices

for each $j \in \mathcal{B} \setminus \{hd\}$:

$\Delta^{(hd \rightarrow j)} = \text{Solder}(E^{(hd)}, E^{(j)})$

$\Delta^{(j \rightarrow hd)} = \text{Solder}(D^{(j)}, D^{(hd)})$

$\Delta^{(hd \rightarrow hd)} :=$ all zeroes // for convenience

func Adjust(X, Δ) // X is a vector of wire labels

for each i do $\tilde{X}[i] = X[i] \oplus \Delta[i, \text{lsb}(X[i])]$

return \tilde{X}

func EvalBucket(\mathcal{B}, X, hd)

for each j in \mathcal{B} :

$\tilde{X}_j = \text{Adjust}(X, \Delta^{(hd \rightarrow j)})$

$Y_j = \text{Adjust}(\text{Eval}(GC^{(j)}, \tilde{X}_j), \Delta^{(j \rightarrow hd)})$

return the majority element of $\{Y_j\}_{j \in \mathcal{B}}$

prot GetInput_{pub}(A, x) // A describes wire labels; x public

P_1 opens commitments of $A|_x^*$; call the result X

P_1 opens commitments of $\tau(A)$

P_2 aborts if $\text{lsb}(X) \neq \tau(A) \oplus x$; else returns X

prot GetInput₁(A, x) // A describes wire labels; P_1 holds x

P_1 opens commitments of $A|_x^*$; return these values

prot GetInput₂(A, x) // A describes wire labels; P_2 holds x

for each position i in A , parties invoke an instance of \mathcal{F}_{ot} :

P_1 uses input $(A[i, A[i, 2]], A[i, 1 \oplus A[i, 2]])$

P_2 uses input x_i

P_2 stores the output as $X[i]$

P_2 returns X

We now describe the main protocol for secure evaluation of Π . We let s denote a statistical security parameter, and T denote an upper bound on the total running time of Π .

1. [Pre-processing phase] **Circuit garbling:** P_1 and P_2 agree on the total number $N = O(sT/\log T)$ of garbled circuits to be generated. Then, for each circuit index $i \in \{1, \dots, N\}$:
 - (a) P_1 chooses random input/output wire label descriptions $E^{(i)}$, $D^{(i)}$ and commits to each of these values component-wise under $\mathcal{F}_{\text{xcom}}$.
 - (b) P_1 computes $GC^{(i)} = \text{Garble}(\tilde{\Pi}, E^{(i)}, D^{(i)})$ and commits to $GC^{(i)}$ under \mathcal{F}_{com} .
2. [Pre-processing phase] **Cut and choose:** P_2 randomly picks a subset S_c of $\{1, \dots, N\}$ of size $N/2$ and sends it to P_1 . S_c will denote the set of check circuits and $S_e = \{1, \dots, N\} \setminus S_c$ will denote the set of evaluation circuits. For check circuit index $i \in S_c$:
 - (a) P_1 opens the commitments of $E^{(i)}$, $D^{(i)}$, and $GC^{(i)}$.
 - (b) P_2 checks that $GC^{(i)} \in \text{Garble}(\tilde{\Pi}, E^{(i)}, D^{(i)})$; if not, P_2 aborts.
3. **Online phase:** For each timestep t :
 - (a) **Bucket creation:** P_2 chooses a random subset of \mathcal{B}_t of S_e of size $\Theta(s/\log T)$ and a random head circuit $hd_t \in \mathcal{B}_t$. P_2 announces them to P_1 . Both parties set $S_e := S_e \setminus \mathcal{B}_t$.
 - (b) **Garbled input: randomness:** P_1 chooses random $r_1 \leftarrow \{0, 1\}^n$, and P_2 chooses random $r_{2,1}, \dots, r_{2,n} \leftarrow \{0, 1\}^n$. P_2 sets

$$\text{rand}_1(X_t) = \text{GetInput}_1(\text{rand}_1(E^{(hd_t)}), r_1)$$

$$\text{rand}_2(X_t) = \text{GetInput}_2(\text{rand}_2(E^{(hd_t)}), r_{2,1} \dots r_{2,n})$$

- (c) **Garbled input: state:** If $t > 1$ then the parties execute:

$$\Delta_{\text{st}} = \text{Solder}(\text{st}(D^{(hd_{t-1})}), \text{st}(E^{(hd_t)}))$$

and P_2 sets $\text{st}(X_t) := \text{Adjust}(\text{st}(Y_{t-1}), \Delta_{\text{st}})$.

Otherwise, in the first timestep, let x_1 and x_2 denote the inputs of P_1 and P_2 , respectively. For input wire labels W , let $\text{st}_1(W)$, $\text{st}_2(W)$, $\text{st}_3(W)$ denote the groups of the internal state wires corresponding to the initial state $x_1 \| x_2 \| 0^n$. To prevent selective abort attacks, we must have P_2 encode his input as n -wise independent shares, as above. P_2 chooses

random $r_{2,1}, \dots, r_{2,n} \in \{0,1\}^n$ such that $\sum_i^n r_{2,i} = x_2$, and sets:³

$$\begin{aligned}\text{st}(X_t) &= \text{GetInput}_1(\text{st}_1(E^{(\text{hd}_t)}), x_1) \\ &\parallel \text{GetInput}_2(\text{st}_2(E^{(\text{hd}_t)}), r_{2,1} \cdots r_{2,n}) \\ &\parallel \text{GetInput}_{\text{pub}}(\text{st}_3(E^{(\text{hd}_t)}), 0^n)\end{aligned}$$

- (d) **Garbled input: memory block:** If the previous instruction $\text{inst}_{t-1} = (\text{READ}, \ell)$ and no previous (WRITE, ℓ) instruction has happened, or if the previous instruction was not a READ, then the parties do $\text{block}(X_t) = \text{GetInput}_{\text{pub}}(\text{block}(E^{(\text{hd}_t)}), 0^n)$. Otherwise, if $\text{inst}_{t-1} = (\text{READ}, \ell)$ and t' is the largest time step with $\text{inst}_{t'} = (\text{WRITE}, \ell)$, then the parties execute:

$$\Delta_{\text{block}} = \text{Solder}(\text{block}(D^{(\text{hd}_{t'})}), \text{block}(E^{(\text{hd}_t)}))$$

Then P_2 sets $\text{block}(X_t) := \text{Adjust}(\text{block}(Y_{t'}), \Delta_{\text{block}})$.

- (e) **Construct bucket:** P_1 and P_2 run subprotocol $\text{MkBucket}(\mathcal{B}_t, \text{hd}_t)$ to assemble the circuits.
(f) **Circuit evaluation:** For each $i \in \mathcal{B}_t$, P_1 opens the commitment to $GC^{(i)}$ and to $\tau(\text{inst}(D^{(i)}))$. P_2 does $Y_t = \text{EvalBucket}(\mathcal{B}_t, X_t, \text{hd}_t)$.
(g) **Output authenticity:** P_2 sends $\tilde{Y} = \text{inst}(Y_t)$ to P_1 . Both parties decode the output $\text{inst}_t = \text{lsb}(\tilde{Y}) \oplus \tau(\text{inst}(D^{(\text{hd}_t)}))$. P_1 aborts if the claimed wire labels \tilde{Y} do not equal the expected wire labels $\text{inst}(D^{(\text{hd}_t)})|_{\text{inst}_t}^*$. If $\text{inst}_t = (\text{HALT}, z)$, then both parties halt with output z .

3.3 Security Proof

Due to page limits, we give only an overview of the simulator \mathcal{S} and security proof. The complete details are deferred to the full version [1].

Assumptions. The security of our protocol relies on the security underlying functionalities, i.e. $\mathcal{F}_{\text{xcom}}, \mathcal{F}_{\text{com}}, \mathcal{F}_{\text{ot}}$, a garbling scheme satisfying properties discussed in Section 2.2, and an ORAM scheme satisfying standard properties discussed in Section 2.1. All the functionalities can be instantiated using standard number theoretic assumptions, and for UC security would be in the CRS model. The garbling scheme can be instantiated using a standard PRF, or using stronger assumptions such as correlation-secure hash functions for taking advantage of

³ We are slightly abusing notation here. More precisely, the parties are evaluating a slightly different circuit $\tilde{\Pi}$ in the first timestep than other timesteps. In the first timestep, it is P_2 's input x_2 that is encoded randomly, whereas in the other steps it is P_2 's share r_2 of the random tape. However, the difference between these circuits is only in the addition of new XOR gates, and only at the input level. When using the Free-XOR optimization, these gates can actually be added after the fact, so the difference is compatible with our pre-processing.

free-XOR. As noted earlier, we do not require the garbling scheme to be adaptively secure, but if so, we can simplify the protocol by not committing to the garbled circuits.

When P_1 is corrupted: The pre-processing phase does not depend on party's inputs, so it is trivial to simulate the behavior of an honest P_2 . However, \mathcal{S} can obtain P_1 's commitments to all circuits and wire labels. Hence, it can determine whether each of these circuits is correct.

In each timestep t of the online phase, \mathcal{S} can abort if an bucket is constructed with a majority of incorrect circuits; this happens with only negligible probability. \mathcal{S} can abort just as an honest P_2 would abort if P_1 cheats in the `Solder`, `GetInput1`, or `GetInputpub` subprotocols. Using a standard argument from [24], \mathcal{S} can also match (up to a negligible difference) the probability of an honest P_2 aborting due to cheating in the `GetInput2` subprotocol. \mathcal{S} can extract P_1 's input x_1 in timestep $t = 1$ by comparing the sent wire labels to the committed wire labels extracted in the offline phase. \mathcal{S} can send x_1 to the ideal functionality and receive the output z . Then \mathcal{S} generates a simulated ORAM memory-access sequence. Each time in step (3g), \mathcal{S} knows all of the relevant wire labels so can send wire labels \tilde{Y} chosen to encode the desired simulated ORAM memory instruction.

When P_2 is corrupted: In the pre-processing phase, \mathcal{S} simulates commit messages from \mathcal{F}_{com} . After receiving S_c from P_2 , it equivocates the opening of the check sets to honestly garbled circuits and wire labels.

In each timestep t of the online phase, \mathcal{S} sends random wire labels in the `GetInput1` and `GetInputpub` subprotocols, and also simulates random wire labels as the output of \mathcal{F}_{ot} in the `GetInput2` subprotocols. These determine the wire labels that are “visible” to P_2 . \mathcal{S} also extracts P_2 's input x_2 from its select bits sent to \mathcal{F}_{ot} . It sends x_2 to the ideal functionality and receives the output z . Then \mathcal{S} generates a simulated ORAM memory-access sequence.

In the `Solder` steps, \mathcal{S} equivocates soldering values chosen to map visible wire labels to their counterparts in other circuits, and chooses random soldering values for the non-visible wire labels. When it is time to open the commitment to the garbled circuit, \mathcal{S} chooses a random set of visible output wire labels and equivocates to a simulated garbled circuit generated using only these visible wire labels. \mathcal{S} also equivocates on the decommitment to the decoding information $\tau(\text{inst}(D^{(i)}))$, chosen so that the visible output wires will decode to the next simulated ORAM memory instruction. Instead of checking P_2 's claimed wire labels in step (3g), the simulator simply aborts if these wire labels are not the pre-determined visible output wire labels.

3.4 Efficiency and Parameter Analysis

In the offline phase, the protocol is dominated by the generation of many garbled circuits, $O(sT / \log T)$ in all. In the full version [1], we describe computation of the exact constant. As an example, for $T = 1$ million, and to achieve statistical security 2^{-40} , it is necessary to generate $10 \cdot T$ circuits in the offline phase.

In the online phase, the protocol is dominated by two factors: the homomorphic decommitments within the **Solder** subprotocol, and the oblivious transfers (in **GetInput**₂) in which P_2 receives garbled inputs. For the former, we require one decommitment for each input and output wire label (to solder that wire to another wire) of the circuit \tilde{H} . Hence the cost in each timestep is proportional to the input/output size of the circuit and the size of the buckets. Continuing our example from above ($T = 10^6$ and $s = 40$), buckets of size 5 are sufficient.

In the full version [1], we additionally discuss parameter settings for when the parties open a different fraction (i.e., not 1/2) of circuits in the cut-and-choose phase. By opening a smaller fraction in the offline phase, we require fewer circuits overall, at the cost of slightly more circuits per timestep (i.e., slightly larger buckets) in the online phase.

We require one oblivious transfer per input bit of P_2 per timestep (independent of the size of buckets). P_2 's input is split in an s -way secret share to assure input-dependent failure probabilities, leading to a total of sn OTs per timestep (where n is the number of random bits required by \tilde{H}). However, online oblivious transfers are inexpensive (requiring only few symmetric-key operations) when instantiated via OT extension [2, 16], where the more expensive “seed OTs” will be done in the pre-processing phase. In Section 5 we suggest further ways to reduce the required number of OTs in the online phase.

Overall, the online overhead of this protocol (compared to the semi-honest setting) is dominated by the bucket size, which is likely at most 5 or 7 for most reasonable settings.

In terms of memory requirements, P_1 must store all pre-processed garbled circuits, and P_2 must store all of their commitments. For each bit of RAM memory, P_1 must store the two wire labels (and their decommitment info) corresponding to that bit, from the last write-time of that memory location. P_2 must store only a single wire label per memory bit.

4 Streaming Cut-and-Choose Protocol

4.1 High-Level Overview

The standard **cut-and-choose approach** is (for evaluating a *single circuit*) for the sender P_1 to garble $O(s)$ copies of the circuit, and receiver P_2 to request half of them to be opened. If all opened circuits are correct, then with overwhelming probability (in s) a majority of the unopened circuits are correct as well.

When trying to apply this methodology to our setting, we face the challenge of feeding past outputs (internal state, memory blocks) into future circuits. Naïvely doing a separate cut-and-choose for each timestep of the RAM program leads to problems when reusing wire labels. Circuits that are opened and checked in time step t must have wire labels independent of past circuits (so that opening these circuits does not leak information about past garbled outputs). Circuits used for evaluation must be garbled with input wire labels *matching* output wire labels of past circuits. But the security of cut and choose demands that P_1 cannot know, at the time of garbling, which circuits will be checked or used for evaluation.

Our alternative is to use a technique suggested by [31] to perform a single cut-and-choose that applies to all timesteps. We make $O(s)$ independent **threads** of execution, where wire labels are directly reused only within a single thread. A cut-and-choose step at the beginning determines whether each *entire thread* is used for checking or evaluation. Importantly, this is done using an oblivious transfer (as in [18, 22]) so that P_1 does not learn the status of the threads.

More concretely, for each thread the parties run an oblivious transfer allowing P_2 to pick up either k_{check} or k_{eval} . Then at each timestep, P_1 sends the garbled circuit but also encrypts the *entire set* of wire labels under k_{check} and encrypts wire labels for only her input under k_{eval} . Hence, in check threads P_2 receives enough information to verify correct garbling of the circuits (including reuse of wire labels — see below), but learns nothing about P_1 's inputs. In evaluation threads, P_2 receives only P_1 's garbled input and the security property of garbled circuits applies. If P_1 behaves incorrectly in a *check thread*, P_2 aborts immediately. Hence, it is not hard to see that P_1 cannot cause a majority of evaluation threads to be faulty while avoiding detection in *all* check threads, except with negligible probability.

Reusing wire labels is fairly straight-forward since it occurs only within a single thread. The next circuit in the thread is simply garbled with input wire labels matching the appropriate output wire labels in the same thread (i.e., the state output of the previous circuit, and possibly the memory-block output wires of an earlier circuit). We point out that P_1 must know the previous memory instruction before garbling the next batch of circuits: if the instruction was (READ, ℓ) , then the next circuit must be garbled with wire labels matching those of the last circuit to write to memory location ℓ . Hence this approach is not compatible with batch pre-processing of garbled circuits.

For enforcing consistency of P_1 's input, we use the approach of [38]⁴, where the very first circuit is augmented to compute a “hiding” universal hash of P_1 's input. For efficiency purposes, the hash is chosen as $M \cdot (x_1 \| r)$, where M is a random binary matrix M of size $s \times (n + 2s + \log s)$ chosen by P_2 . We prevent input-dependent abort based on P_2 's input using the XOR-tree approach of [24], also used in the previous protocol.

We ensure authenticity of the output for P_1 using an approach suggested in [30]. Namely, wire labels corresponding to the same output wire and truth value are used to encrypt a random “output authenticity” key. Hence P_2 can compute these output keys only for the circuit's true output. P_2 is not given the information required for checking these ciphertexts until after he *commits* to the output keys. At the time of committing, he cannot guess complementary output keys, but he does not actually open the commitment until he receives the checking information and is satisfied with the check circuits.

The adaptation of the input-recovery technique of Lindell [23] is more involved and hence we discuss it separately in Section 4.5.

⁴ Although our protocol is also compatible with the solution of [30].

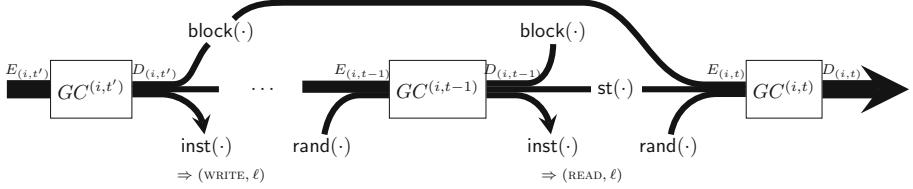


Fig. 4. Wire-label reuse within a single thread i , in the streaming cut-and-choose protocol

4.2 Detailed Protocol Description

We now describe the streaming cut-and-choose protocol for secure evaluation of Π , the ORAM program to be computed. Recall that $\tilde{\Pi}(\text{st}, \text{block}, \text{inp}_1, \text{inp}_{2,1}, \dots, \text{inp}_{2,n}) = \Pi(\text{st}, \text{block}, \text{inp}_1 \oplus_i \text{inp}_{2,i})$. We let s denote a statistical security parameter parameter, and T denote an upper bound on the total running time of Π . Here, we describe the majority-evaluation variant of the protocol and discuss how to integrate the input-recovery technique in Section 4.5.

1. **Cut-and-choose.** The parties agree on $S = O(s)$, the number of threads (see discussion below). P_2 chooses a random string $b \leftarrow \{0, 1\}^S$. Looking ahead, thread i will be a check thread if $b_i = 0$ and an evaluation thread if $b_i = 1$.

For each $i \in \{1, \dots, S\}$, P_1 chooses two symmetric encryption keys $k_{(i,\text{check})}$ and $k_{(i,\text{eval})}$. The parties invoke an instance of \mathcal{F}_{ot} with P_2 providing input b_i and P_1 providing input $(k_{(i,\text{check})}, k_{(i,\text{eval})})$.

2. **RAM evaluation.** For each timestep t , the following are done in parallel for each thread $i \in \{1, \dots, S\}$:

- (a) **Wire label selection.** P_1 determines the input wire labels $E_{(t,i)}$ for garbled circuit $GC_{(t,i)}$ as follows. If $t = 1$, these wire labels are chosen uniformly. Otherwise, we set $\text{st}(E_{(t,i)}) = \text{st}(D_{(t-1,i)})$ and choose $\text{rand}_1(E_{(t,i)})$ and $\text{rand}_2(E_{(t,i)})$ uniformly. If the previous instruction $\text{inst}_{t-1} = (\text{READ}, \ell)$ and no previous (WRITE, ℓ) instruction has happened, or if the previous instruction was not a READ, then P_1 chooses $\text{block}(E_{(t,i)})$ uniformly at random. Otherwise, we set $\text{block}(E_{(t,i)}) = \text{block}(D_{(t',i)})$, where t' is the last instruction that wrote to memory location ℓ .
- (b) **Input selection.** Parties choose shares of the randomness required for $\tilde{\Pi}$: P_1 chooses $r_1 \leftarrow \{0, 1\}^n$, and P_2 chooses $r_{2,1}, \dots, r_{2,n} \leftarrow \{0, 1\}^n$.
- (c) **P_1 's garbled input transfer.** P_1 sends the following wire labels, encrypted under $k_{(i,\text{eval})}$:

$$\begin{aligned} & \text{st}_1(E_{(t,i)})|_{x_1}^* \text{ if } t = 1 \\ & \text{rand}_1(E_{(t,i)})|_{r_1}^* \end{aligned}$$

The following additional wire labels are also sent in the clear:

$$\begin{aligned} & \text{st}_3(E_{(t,i)})|_{0^n}^* \text{ if } t = 1 \\ & \text{block}(E_{(t,i)})|_{0^n}^* \text{ if WRITE or uninitialized READ} \end{aligned}$$

- (d) **P_2 's garbled input transfer.** P_2 obtains garbled inputs via calls to OT. To guarantee that P_2 uses the same input in all threads, we use a single OT across all threads for each input bit of P_2 . For each input bit, P_1 provides the true and false wire labels for all threads as input to \mathcal{F}_{ot} , and P_2 provides his input bit as the OT select bit.

Note that P_2 's inputs consist of the strings $r_{2,1}, \dots, r_{2,n}$ as well as the string x_2 for the case of $t = 1$.

- (e) **Input consistency.** If $t = 1$, then P_2 sends a random $s \times (n + 2s + \log s)$ binary matrix M to P_1 . P_1 chooses random input $r \in \{0,1\}^{2s+\log s}$, and augments the circuit for $\tilde{\Pi}$ with a subcircuit for computing $M \cdot (x_1 \| r)$.
- (f) **Circuit garbling.** P_1 chooses output wire labels $D_{(t,i)}$ at random and does $GC^{(t,i)} = \text{Garble}(\tilde{\Pi}, E_{(t,i)}, D_{(t,i)})$, where in the first timestep, $\tilde{\Pi}$ also contains the additional subcircuit described above. P_1 sends $GC^{(t,i)}$ to P_2 as well as $\tau(\text{inst}(D_{(t,i)}))$.

In addition, P_1 chooses a random Δ_t for this time-step and for each inst-output bit j , he chooses random strings $w_{(t,j,0)}$ and $w_{(t,j,1)}$ (the same across all threads) to be used for output authenticity, such that $w_{(t,j,0)} \oplus w_{(t,j,1)} = \Delta_t$. For each thread i , output wire j and select bit b corresponding to truth value b' , let $v_{i,j,b}$ denote the corresponding wire label. P_1 computes $c_{i,j,b} = \text{Enc}_{v_{i,j,b}}(w_{(t,j,b')})$ and $h_{i,j,b} = H(c_{i,j,b})$, where H is a 2-Universal hash function. P_1 sends $h_{i,j,b}$ in the clear and sends $c_{i,j,b}$ encrypted under $k_{(\text{eval},i)}$.

- (g) **Garbled input collection.** If thread i is an evaluation thread, then P_2 assembles input wire labels $X_{(t,i)}$ for $GC^{(t,i)}$ as follows:
 P_2 uses $k_{(\text{eval},i)}$ to decrypt wire labels sent by P_1 . Along with the wire labels sent in the clear and those obtained via OTs in GetInput_2 , these wire labels will comprise $\text{rand}(X_{(t,i)})$; $\text{block}(X_{(t,i)})$ in the case of a WRITE or uninitialized READ; and $\text{st}(X_{(t,i)})$ when $t = 1$.

Other input wire labels are obtained via:

$$\begin{aligned}\text{st}(X_{(t,i)}) &= \text{st}(Y_{(t-1,i)}) \\ \text{block}(X_{(t,i)}) &= \text{block}(Y_{(t',i)})\end{aligned}$$

where t' is the last write time of the appropriate memory location, and Y denote the output wire labels that P_2 obtained during previous evaluations.

- (h) **Evaluate and commit to output.** If thread i is an eval thread, then P_2 evaluates the circuit via $Y_{(t,i)} = \text{Eval}(GC^{(t,i)}, X_{(t,i)})$ and decodes the output $\text{inst}_{(t,i)} = \text{lsb}(Y_{(t,i)}) \oplus \tau(D_{(t,i)})$. He sets $\text{inst}_t = \text{majority}_i\{\text{inst}_{(t,i)}\}$. For each inst-output wire label j , P_2 decrypts the corresponding ciphertext $c_{i,j,b}$, then takes w'_j to be the majority result across all threads i . P_2 commits to w'_j .

If $t = 1$, then P_2 verifies that the output of the auxiliary function $M \cdot (x_1 \| r)$ is identical to that of all other threads; if not, he aborts.

- (i) **Checking the check threads.** P_1 sends $\text{Enc}_{k_{(i,\text{check})}}(seed_{(t,i)})$ to P_2 , where $seed_{(t,i)}$ is the randomness used in the call to Garble . Then if thread i

is a check thread, P_2 checks the correctness of $GC^{(t,i)}$ as follows. By induction, P_2 knows all the previous wire labels in thread i , so can use $seed_{(t,i)}$ to verify that $GC^{(t,i)}$ is garbled using the correct outputs. In doing so, P_2 learns all of the output wire labels for $GC^{(t,i)}$ as well. P_2 checks that the wire labels sent by P_1 in the clear are as specified in the protocol, and that the $c_{i,j,b}$ ciphertexts and $h_{i,j,b}$ are correct and consistent. He also decrypts $c_{i,j,b}$ for $b \in \{0, 1\}$ with the corresponding output label to recover $w'_{(t,j,b)}$ and checks that $w'_{(t,j,0)} \oplus w'_{(t,j,1)}$ is the same for all j . Finally, P_2 checks that the wire labels obtained via OT in GetInput_2 are the correct wire labels encoding P_2 's provided input. If any of these checks fail, then P_2 aborts immediately.

- (j) **Output verification.** P_2 opens the commitments to values w'_j and P_1 uses them to decode the output inst_t . If a value w'_j does not match one of $w_{(t,j,0)}$ or $w_{(t,j,1)}$, then P_1 aborts.

4.3 Security Proof

Again we only give a brief overview of the simulator, with the details deferred to the full version [1].

The security of the protocol relies on functionalities \mathcal{F}_{com} , \mathcal{F}_{ot} which can both be instantiated under number theoretic assumptions in the CRS model, a secure garbling scheme and an ORAM scheme satisfying standard properties discussed earlier. More efficiency can be obtained using RO or correlation-secure hash functions, to take advantage of the free-XOR technique for garbling (and faster input-consistency checks), or the use of fast OT extension techniques.

When P_1 is corrupt: In the cut-and-choose step, the simulator \mathcal{S} extracts both encryption keys $k_{(i,\text{eval})}$ and $k_{(i,\text{check})}$. Just as P_2 , the simulator designates half of the threads to be check threads and half to be eval threads, and aborts if a check thread is ever found to be incorrect. However, the simulator can perform the same check for all threads, and keeps track of which eval threads are correct. A standard argument shows that if all check threads are correct, then a majority of eval threads are also correct, except with negligible probability. Without loss of generality, we can have \mathcal{S} abort if this condition is ever violated.

Knowing both encryption keys, \mathcal{S} can associate P_1 's input wire labels with truth values (at least in the correct threads). If P_1 provides disagreeing inputs x_1 among the correct eval threads, then \mathcal{S} aborts, which is negligibly close to P_2 's abort probability (via the argument regarding the input-consistency of [38]). Otherwise, this determines P_1 's input x_1 which \mathcal{S} sends to the ideal functionality, receiving output z in return. \mathcal{S} generates a simulated ORAM memory access pattern.

In the output commitment step, \mathcal{S} simulates a commit message. Then after the check phase, \mathcal{S} learns all of the output-authenticity keys. So \mathcal{S} simply equivocates the opening of the output keys to be the ones encoding the next ORAM memory instruction.

When P_2 is corrupt: In the cut-and-choose phase, \mathcal{S} extracts P_2 's selection of check threads and eval threads. In check threads, \mathcal{S} always sends correctly

generated garbled circuits, following the protocol specification and generates dummy ciphertexts for the encryptions under $k_{(i,\text{eval})}$. Hence, these threads can be simulated independently of P_1 's input.

In each eval thread, \mathcal{S} maintains visible input/output wire labels for each circuit, choosing new output wire labels at random. \mathcal{S} ensures that P_2 picks up these wire labels in the input collection step. \mathcal{S} also extracts P_2 's input x_2 in this phase, from its select bit inputs to \mathcal{F}_{ot} . \mathcal{S} sends x_2 to the ideal functionality and receives output z . Then \mathcal{S} generates a simulated ORAM memory access pattern.

At each timestep, for each eval thread, \mathcal{S} generates a simulated garbled circuit, using the appropriate visible input/output wire labels. It fixes the decoding information τ so that the visible output wire labels will decode to the appropriate ORAM instruction. In the output reveal step, \mathcal{S} aborts if P_2 does not open its commitment to the expected output keys. Indeed, P_2 's view in the simulation is independent of the complementary output keys.

4.4 Efficiency and Parameter Analysis

At each timestep, the protocol is dominated by the generation of S garbled circuits (where S is the number of threads) as well as the oblivious transfers for P_2 's inputs. As before, using OT extension as well as the optimizations discussed in Section 5, the cost of the oblivious transfers can be significantly minimized. Other costs in the protocol include simple commitments and symmetric encryptions, again proportional to the number of threads. Hence the major computational overhead is simply the number of threads. An important advantage of this protocol is that we avoid the soldering and the “expensive” xor-homomorphic commitments needed for input/outputs of each circuit in our batching solution. On the other hand, this protocol always require $O(s)$ garbled circuit executions regardless of the size of the RAM computation, while as discussed earlier, our batching protocol can require significantly less garbled circuit execution when the running time T is large. The choice of which protocol to use would then depend on the running time of the RAM computation, the input/output size of the next-instruction circuits as well as practical efficiency of xor-homomorphic commitment schemes in the future.

Compared to our other protocol, this one has a milder memory requirement. Garbled circuits are generated on the fly and can be discarded after they are used, with the exception of the wire labels that encode memory values. P_1 must remember $2S$ wire labels per bit of memory (although in Section 5 we discuss a way to significantly reduce this requirement). P_2 must remember between S and $2S$ wire labels per bit of memory (1 wire label for evaluation threads, 2 wire labels for check threads).

Using the standard techniques described above, we require $S \approx 3s$ threads to achieve statistical security of 2^{-s} . Recently, techniques have been developed [23] for the SFE setting that require only s circuits for security 2^{-s} (concretely, s is typically taken to be 40). We now discuss the feasibility of adapting these techniques to our protocol:

4.5 Integrating Cheating Recovery

The idea of [23] is to provide a mechanism that would detect inconsistency in the output wire labels encoding the final output of the computation. If P_2 receives output wire labels for two threads encoding disparate values, then a secondary computation allows him to recover P_1 's input (and hence compute the function himself). This technique reduces the number of circuits necessary by a factor of 3 since we only need a single honest thread among the set of evaluated threads (as opposed to a majority). We refer the reader to [23] for more details. We point out that in some settings, recovering P_1 's input may not be enough. Rather, if P_2 is to perform the entire computation on his own in the case of a cheating P_1 , then he also needs to know the contents of the RAM memory!

Cheating recovery at each timestep. It is possible to adapt this approach to our setting, by performing an input-recovery computation at the end of each timestep. But this would be very costly, since each input-recovery computation is a maliciously secure 2PC that requires expensive input-consistency checks for both party's inputs, something we worked hard to avoid for the state/memory bits. Furthermore, each cheating-recovery garbled circuit contains non-XOR gates that need to be garbled/evaluated 3s times at each timestep. These additional costs can become a bottleneck in the computation specially when the next-instruction circuit is small.

Cheating recovery at the end. It is natural to consider delaying the input-recovery computation until the last timestep, and only perform it once. If two of the threads in the final timestep (which also computes the final output of computation) output different values, the evaluator recovers the garbler's input. Unfortunately, however, this approach is not secure. In particular, a malicious P_1 can cheat in an intermediate timestep by garbling one or more incorrect circuits. This could either lead to two or more valid memory instruction/location outputs, or no valid outputs at all. It could also lead to a premature "halt" instruction. In either case, P_2 cannot yet abort since that would leak extra information about his private input. He also cannot continue with the computation because he needs to provide P_1 with the next instruction along with proof of its authenticity (i.e. the corresponding garbled labels) but that would reveal information about his input.

We now describe a solution that avoids the difficulties mentioned above and at the same time eliminates the need for input-consistency checks or garbling/evaluating non-XOR gates at each timestep. In particular, we delay the "proof of authenticity" by P_2 for all the memory instructions until after the last timestep. Whenever P_2 detects cheating by P_1 (i.e. more than two valid memory instructions), instead of aborting, he pretends that the computation is going as planned and sends "dummy memory operations" to P_1 but does not (and cannot) prove the authenticity of the corresponding wire labels yet. For modern tree-based ORAM constructions ([7, 40], etc) the memory access pattern is always uniform, so it is easy for P_2 to switch from reporting the real memory

access pattern to a simulated one. Note that in step (h) of the protocol, P_2 no longer needs to commit to the majority w'_j . As a result, step (j) of the protocol will be obsolete. Instead, in step (h), P_2 sends the `instt` in plaintext. This instruction is the single valid instruction he has recovered or a dummy instruction (if P_2 has attempted to cheat).

After the evaluation of the final timestep, we perform a fully secure 2PC for an input-recovery circuit that has two main components. The first one checks if P_1 has cheated. If he has, it reveals P_1 's input to P_2 . The second one checks the proofs of authenticity of the `inst` instructions P_2 reveals in all timesteps and signals to P_1 to abort if the proof fails.

First cheating recovery, then opening the check circuits. For this cheating recovery method to work, we perform the evaluation steps (step (h)) for all timesteps first (at this stage, P_2 only learns the labels for the final output but not the actual value), then perform the cheating recovery as described above, and finally perform all the checks (step (i)) for all time-steps.

We now describe the cheating recovery circuit which consists of two main components in more detail.

- The first component is similar to the original cheating recovery circuit of [23]. P_2 's input is the XOR of two valid output authenticity labels for a wire j at step t for which he has detected cheating (if there is more than one instance of cheating he can use the first occurrence). Lets denote the output authenticity labels for j th bit of $\text{block}(Y_{(t,i)})$ at time-step t with $w_{(t,j,b)}$, $b \in \{0, 1\}$. Then P_2 will input $w_{(t,j,0)} \oplus w_{(t,j,1)}$ to the circuit. If there is no cheating, he inputs garbage. Notice that $w_{(t,j,0)} \oplus w_{(t,j,1)} = \Delta_t$ for valid output authenticity values, as described in the protocol (note that we assume that all output authenticity labels in timestep t use the same offset Δ_t).

P_1 inputs his input x_1 . He also hardcodes Δ_t . For timestep t (as shown in Figure 5) the circuit compares P_2 's input against the hardcoded Δ_t . If P_2 's input is the same as the Δ_t , cheating is detected and the circuit outputs 1. To check that P_2 's input is the same as at least one of the hard-coded Δ s, in the circuit of Figure 6 we compute the OR of all these outputs. Thus, if the output of this circuit is 1, it means that P_1 has cheated in at least one timestep.

To reveal P_1 's input, we compute the AND of output of circuit of Figure 6 with each bit of P_1 's input as depicted in Figure 7. This concludes the description of the first component for cheating recovery.

- In the second component, we check the authenticity of the memory instructions P_2 provided in all timesteps. In particular, he provides the hash of concatenation of all output authentication labels he obtained during the evaluation corresponding to `inst` in all timesteps (P_2 uses dummy labels if he does not have valid ones due to P_1 's cheating), while P_1 does the same based on the plaintext instructions he received from P_2 and the labels which he knows. The circuit then outputs 1 if the two hash values match. The circuit

structure is therefore identical to that of Figure 5, but the inputs are the hash values. An output of 0 would mean that P_2 does not have a valid proof of authenticity.

As shown in the final circuit of Figure 7 then, if P_1 was not already caught cheating in the previous step, and P_2 's proof of authenticity fails, the circuit outputs a 1 to signal an abort to P_1 . This is a crucial condition, i.e., it is important to ensure P_1 did not cheat (the output of circuit of Figure 6) before accusing P_2 of cheating, since in case of cheating by P_1 say in timestep t , P_2 may be able to prove authenticity of the instructions for timestep t or later.

Efficiency: Following the techniques of [23], all the gates of Figures 5, and 6 can be garbled using non-cryptographic operations (XORs) and only the circuit of Figure 7 has non-XOR gates. More precisely it requires $|x_1|$ ANDs and a NOT gate.

Of course, the final circuit will be evaluated using a basic maliciously secure 2PC. Thus, we need to add a factor of $3s$ to the above numbers which results in garbling a total of $3s(|x_1| + 1)$ non-XOR gates which is at most $12s(|x_1| + 1)$ symmetric operations.

The input consistency checks are also done for P_1 's input x_1 and P_2 's input which is a proof of cheating of length $|\Delta|$ and a proof of authenticity which is the output of a hash function (both are in the order of the computational security parameter). We stress that the gain is significant since both the malicious 2PC and the input consistency checks are only done once at the end.

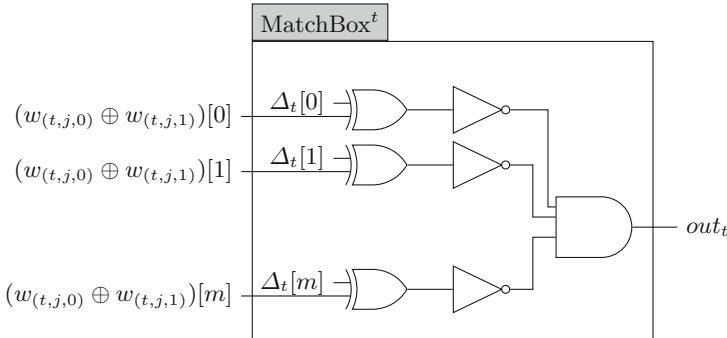


Fig. 5. Cheating recovery component 1: MatchBox. Where $\Delta_t[i]$ denotes the i th bit of Δ_t and $m = |\Delta_t|$.

5 Optimizations

Here we present a collection of further optimizations compatible with our 2PC protocols:

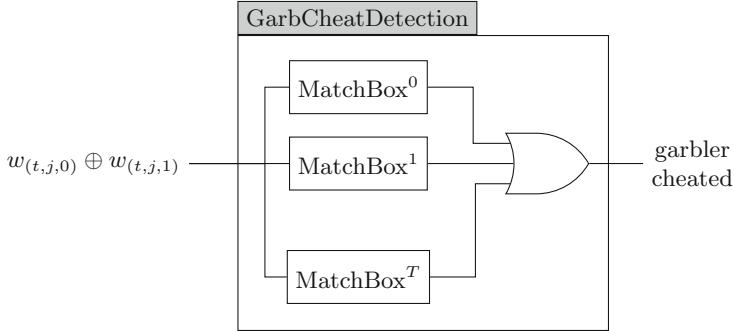


Fig. 6. Cheating Recovery component 1: Garbler Cheating Detection

5.1 Hide only the Input-Dependent Behavior

Systems like SCVM [27] use static program analysis to “factor out” as much input-independent program flow as possible from a RAM computation, leaving significantly less residual computation that requires protection from the 2PC mechanisms.

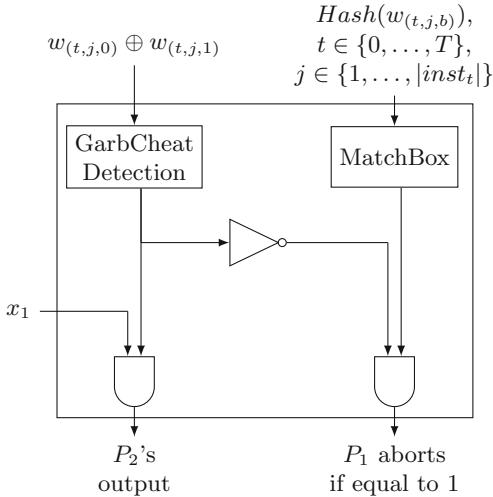
The backend protocol currently implemented by SCVM achieves security only against semi-honest adversaries. However, our protocols are also compatible with their RAM-level optimizations, which we discuss in more detail:

Special-purpose circuits. For notational simplicity, we have described our RAM programs via a *single* circuit Π that evaluates each timestep. Then Π must contain subcircuits for every low-level instruction (addition, multiplication, etc) that may ever be needed by this RAM program.

Instruction-trace obliviousness means that the choice of low-level instruction (e.g., addition, multiplication) performed at each time t does not depend on private input. The SCVM system can compile a RAM program into an instruction-trace-oblivious one (though one does not need full instruction-trace obliviousness to achieve an efficiency gain in 2PC protocols). For RAM programs with this property, we need only evaluate an (presumably much smaller) instruction-specific circuit Π_t at each timestep t .

It is quite straight-forward to evaluate different circuits at different timesteps in our cut-and-choose protocol of Section 4. For the batching protocol of Section 3, enough instruction-specific circuits must be generated in the pre-processing phase to ensure a majority of correct circuits in each bucket. However, we point out that buckets at different timesteps could certainly be different sizes! One particularly interesting use-case would involve a very aggressive pre-processing of the circuits involved in the ORAM construction (i.e., the logic translating logical memory accesses to physical accesses), since these will dominate the computation and do not depend on the functionality being computed.⁵ The bucket size / replication factor for these timesteps could be very low (say, 5), while the less-aggressively pre-processed instructions could have larger buckets. In this case,

⁵ Such pre-processing yields an instance of *commodity-based MPC* [3].

**Fig. 7.** Final Circuit

the plain-RAM internal state could be kept separate from the ORAM-specific internal state, and only fed into the appropriate circuits.

Along similar lines, we have for simplicity described RAM programs that require a random input tape at each timestep. This randomness leads to oblivious transfers within the protocol. However, if it is known to both parties that a particular instruction does not require randomness, then these OTs are not needed. For example, deterministic algorithms require randomness only for the ORAM mechanism. Concretely, tree-based ORAM constructions [7, 39, 40] require only a small amount of randomness and at input-independent steps.

Memory-trace obliviousness. Due to their general-purpose nature, ORAM constructions protect *all* memory accesses, even those that may already be input-independent (for example, sequential iteration over an array). One key feature of SCVM is detecting which memory accesses are already input-independent and not applying ORAM to them. Of course, such optimizations to a RAM program would yield benefit to our protocols as well.

5.2 Reusing Memory

We have described our protocols in terms of a single RAM computation on an initially empty memory. However, one of the “killer applications” of RAM computations is that, after an initial quasi-linear-time ORAM initialization of memory, future computations can use time sublinear in the total size of data (something that is impossible with circuits). This requires an ORAM-initialized memory to be reused repeatedly, as in [13].

Our protocols are compatible with reusing garbled memory. In particular, this can be viewed as a single RAM computation computing a reactive functionality (one that takes inputs and gives outputs repeatedly).

5.3 Other Protocol Optimizations

Storage requirements for RAM memory. In our cut-and-choose protocol, P_1 chooses random wire labels to encode bits of memory, and then has to remember these wire labels when garbling later circuits that read from those locations. As an optimization, P_1 could instead choose wire labels via $F_k(t, j, i, b)$, where F is a suitable PRF, t is the timestep in which the data was written, j is the index of a thread, i is the bit-offset within the data block, and b is the truth value. Since memory *locations* are computed at run-time, P_1 cannot include the memory location in the computation of these wire labels. Hence, P_1 will still need to remember, for each memory location ℓ , the last timestep t at which location ℓ was written.

Adaptive garbling. In the batching protocol, P_1 must commit to the garbled circuits and reveal them only after P_2 obtains the garbled inputs. This is due to a subtle issue of (non)adaptivity in standard security definitions of garbled circuits; see [4] for a detailed discussion. These commitments could be avoided by using an adaptively-secure garbling scheme.

Online/offline tradeoff. For simplicity we described our online/offline protocol in which P_1 generates many garbled circuits and P_2 opens exactly half of them. Lindell and Riva [26] also follow a similar approach of generating many circuits in an offline phase and assigning the remainder to random buckets; they also point out that changing the fraction of opened circuits results in different tradeoffs between the amount of circuits used in the online and offline phases. For example, checking 20% of circuits results in fewer circuits overall (i.e., fewer generated in the offline phase) but larger buckets (in our setting, more garbled circuits per timestep in the online phase).

References

1. Afshar, A., Hu, Z., Mohassel, P., Rosulek, M.: How to efficiently evaluate ram programs with malicious security. Cryptology ePrint Archive, Report 2014/759 (2014). <http://eprint.iacr.org/>
2. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: Sadeghi, et al. (eds.) [35], pp. 535–548
3. Beaver, D.: Commodity-based cryptography (extended abstract). In: 29th Annual ACM Symposium on Theory of Computing, pp. 446–455. ACM Press, May 1997
4. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 134–153. Springer, Heidelberg (2012)
5. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, et al. (eds.) [43], pp. 784–796
6. Chen, J., Wee, H.: Fully, (almost) tightly secure IBE and dual system groups. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 435–460. Springer, Heidelberg (2013)

7. Chung, K.-M., Pass, R.: A simple ORAM. Cryptology ePrint Archive, Report 2013/243 (2013). <http://eprint.iacr.org/2013/243>
8. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, Canetti. (eds.) [36], pp. 643–662
9. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B., Nordholt, P.S., Orlandi, C.: MiniLEGO: efficient secure two-party computation from general assumptions. In: Johansson, Nguyen (eds.) [17], pp. 537–556
10. Gentry, C., Halevi, S., Lu, S., Ostrovsky, R., Raykova, M., Wichs, D.: Garbled RAM revisited. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 405–422. Springer, Heidelberg (2014)
11. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing, pp. 218–229. ACM Press, May 1987
12. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)
13. Gordon, S.D., Katz, J., Kolesnikov, V., Krell, F., Malkin, T., Raykova, M., Vahlis, Y.: Secure two-party computation in sublinear (amortized) time. In: Yu, et al. (eds.) [43], pp. 513–524
14. Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Canetti, Garay (eds.) [6], pp. 18–35
15. Huang, Y., Katz, J., Kolesnikov, V., Kumaresan, R., Malozemoff, A.J.: Amortizing garbled circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 458–475. Springer, Heidelberg (2014)
16. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
17. Johansson, T., Nguyen, P.Q. (eds.): EUROCRYPT 2013. LNCS, vol. 7881. Springer, Heidelberg (2013)
18. Kamara, S., Mohassel, P., Riva, B.: Salus: a system for server-aided secure function evaluation. In: Yu, et al. (eds.) [43], pp. 797–808
19. Keller, M., Scholl, P.: Efficient, oblivious data structures for MPC. Cryptology ePrint Archive, Report 2014/137 (2014). <http://eprint.iacr.org/>
20. Kiraz, M., Schoenmakers, B.: A protocol issue for the malicious case of yao’s garbled circuit construction. In: 27th Symposium on Information Theory in the Benelux, pp. 283–290 (2006)
21. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórssón, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
22. Kreuter, B., Shelat, A., Shen, C.-H.: Billion-gate secure computation with malicious adversaries. In: Proceedings of the 21st USENIX Conference on Security Symposium, p. 14. USENIX Association (2012)
23. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, Garay (eds.) [6], pp. 1–17
24. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
25. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer, Heidelberg (2011)

26. Lindell, Y., Riva, B.: Cut-and-Choose yao-based secure computation in the online/offline and batch settings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 476–494. Springer, Heidelberg (2014)
27. Liu, C., Huang, Y., Shi, E., Katz, J., Hicks, M.: Automating efficient RAM-model secure computation. In: Proceedings of the IEEE Symposium on Security and Privacy (Oakland), May 2014
28. Lu, S., Ostrovsky, R.: How to garble RAM programs. In: Johansson, Nguyen (eds.) [17], pp. 719–734
29. Mohassel, P., Franklin, M.K.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006)
30. Mohassel, P., Riva, B.: Garbled circuits checking garbled circuits: more efficient and secure two-party computation. In: Canetti, Garay (eds.) [6], pp. 36–53
31. Mood, B., Gupta, D., Feigenbaum, J., Butler, K.: Reuse it or lose it: more efficient secure computation through reuse of encrypted values. In: ACM CCS (2014)
32. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, Canetti (eds.) [36], pp. 681–700
33. Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer, Heidelberg (2009)
34. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
35. Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.): ACM CCS 13: 20th Conference on Computer and Communications Security. ACM Press, November 2013
36. Safavi-Naini, R., Canetti, R. (eds.): CRYPTO 2012. LNCS, vol. 7417. Springer, Heidelberg (2012)
37. Shelat, A., Shen, C.: Two-Output secure computation with malicious adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer, Heidelberg (2011)
38. Shelat, A., Shen, C.-H.: Fast two-party secure computation with minimal assumptions. In: Sadeghi, et al. (eds.) [35], pp. 523–534
39. Shi, E., Chan, T.-H.H., Stefanov, E., Li, M.: Oblivious RAM with $O((\log N)^3)$ worst-case cost. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 197–214. Springer, Heidelberg (2011)
40. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C.W., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. In: Sadeghi, et al. (eds.) [35], pp. 299–310
41. Yao, A.C.-C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, pp. 160–164. IEEE Computer Society Press, November 1982
42. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, pp. 162–167. IEEE Computer Society Press, October 1986
43. Yu, T., Danezis, G., Gligor, V.D. (eds.): ACM CCS 12: 19th Conference on Computer and Communications Security. ACM Press, October 2012
44. Zahur, S.: Obliv-c: a lightweight compiler for data-oblivious computation. In: Workshop on Applied Multi-Party Computation. Microsoft Research, Redmond (2014)

Symmetric Cryptanalysis III

Cube Attacks and Cube-Attack-Like Cryptanalysis on the Round-Reduced Keccak Sponge Function

Itai Dinur^{1(✉)}, Paweł Morawiecki^{2,3}, Josef Pieprzyk⁴, Marian Srebrny^{2,3},
and Michał Straus³

¹ Computer Science Department, École Normale Supérieure, Paris, France
itai.dinur@ens.fr

² Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
pawelm@wsh-kielce.edu.pl, marians@ipipan.waw.pl

³ Section of Informatics, University of Commerce, Kielce, Poland
foldax@gmail.com

⁴ Queensland University of Technology, Brisbane, Australia
josef.pieprzyk@qut.edu.au

Abstract. In this paper, we comprehensively study the resistance of keyed variants of SHA-3 (Keccak) against algebraic attacks. This analysis covers a wide range of key recovery, MAC forgery and other types of attacks, breaking up to 9 rounds (out of the full 24) of the Keccak internal permutation much faster than exhaustive search. Moreover, some of our attacks on the 6-round Keccak are completely practical and were verified on a desktop PC. Our methods combine cube attacks (an algebraic key recovery attack) and related algebraic techniques with structural analysis of the Keccak permutation. These techniques should be useful in future cryptanalysis of Keccak and similar designs.

Although our attacks break more rounds than previously published techniques, the security margin of Keccak remains large. For Keyak – the Keccak-based authenticated encryption scheme – the nominal number of rounds is 12 and therefore its security margin is smaller (although still sufficient).

Keywords: Keccak · SHA-3 · Sponge function · Cube attack

1 Introduction

In 2007, the U.S. National Institute of Standards and Technology (NIST) announced a public contest aiming at the selection of a new standard for a cryptographic hash function. In 2012, after 5 years of intensive scrutiny, the winner was selected. The new SHA-3 standard is the Keccak hash function [8].

As a new standard, Keccak will be widely deployed, and therefore understanding its security is crucial. Indeed, the hash function and its internal permutation

have been extensively analysed in many research papers [2, 6, 11–13, 17, 21, 22]. However, most papers focused on key-less variants of the hash function, whereas Keccak is actually a family of sponge functions [7] and can also be used in keyed modes. In such modes, the sponge function can generate an infinite bit stream, making it suitable to work as a stream cipher or as a pseudorandom bit generator. Furthermore, the sponge function can be used as a building block for message authentication codes (MACs) and authenticated encryption (AE) schemes as described in [10].

In this paper, we aim at filling the gap in the security analysis of Keccak by analysing keyed modes of its variants, in which the number of rounds of the internal permutation is reduced from the full 24. We analyse concrete instances of stream ciphers, MACs and AE schemes based on the Keccak permutation and investigate their resistance against strong key recovery attacks and weaker types of attacks (such as a MAC forgery). The stream cipher and MAC constructions we analyse are instantiated according to the design strategy of [10], which describes methods for building such ciphers from the sponge function. The AE scheme we analyse is called Lake Keyak — a recently proposed authenticated encryption scheme, submitted by the Keccak designers to the CAESAR competition [1] for authenticated encryption.

All of our attacks are closely related to high order differential cryptanalysis [20], which used, for the first time, high order derivatives in cryptanalysis of ciphers with low algebraic degree. As the degree of a round of the Keccak internal permutation is only 2, it makes the round-reduced Keccak a natural target for these types of attacks.

Our analysis is divided into three parts. First, we investigate the security of keyed modes of Keccak against cube attacks — an algebraic key recovery technique introduced in [14]. This attack was previously applied to the 4-round Keccak in [21], and we show how to break up to 6 rounds of some Keccak variants. As our optimized cube attacks have very low complexity, they were implemented and verified on a desktop PC.

In the second part of our analysis, we study the security of keyed modes of the round-reduced Keccak against the cube testers [3]. Unlike the cube attack, cube testers do not recover the secret key, but allow to predict outputs of the scheme for previously unseen inputs, giving rise to other types of attacks (such as MAC forgery). Here, we present theoretical attacks on up to 9 rounds of keyed Keccak variants. Although the attack complexities are impractical, they are significantly faster than generic attacks.

Finally, we reconsider key recovery attacks, and show how to recover the key for 7 rounds of Keccak, much faster than exhaustive search. Table 1 summarizes our attacks on keyed modes of Keccak.

The attacks developed in this paper are described with an increasing degree of sophistication. We start by describing rather simple (yet effective) techniques, such as partial inversion of the internal non-linear mapping of Keccak. Then, our more complex methods exploit additional structural properties of the Keccak permutation (namely, the limited diffusion of its internal mappings) in order to optimize cube testers. Finally, we devise a new key recovery method which also

Table 1. Parameters and complexities of our attacks

Mode	Rounds	Type of Attack	Key size	Time	Data	Reference
MAC	5	Key Recovery	128	2^{36}	2^{35}	Sect. 4
MAC	6	Key Recovery	128	2^{66}	2^{64}	Sect. 6
MAC	7	Key Recovery	128	2^{97}	2^{64}	Sect. 6
MAC	7	Forgery	128	2^{65}	2^{65}	Sect. 5
MAC	8	Forgery	256	2^{129}	2^{129}	Sect. 5
AE (Keyak)	6	Key Recovery (nonce respected)	128	2^{37}	2^{36}	Sect. 4
AE (Keyak)	7	Key Recovery (nonce reused)	128	2^{76}	2^{75}	Sect. 6
AE (Keyak)	7	Forgery (nonce reused)	128	2^{65}	2^{65}	Sect. 5
Stream Cipher	6	Key Recovery	128	2^{37}	2^{36}	Sect. 4
Stream Cipher	8	Keystream Prediction	256	2^{128}	2^{128}	Sect. 5
Stream Cipher	9	Keystream Prediction	512	2^{256}	2^{256}	Sect. 5

exploits the limited diffusion properties. Yet, the key recovery technique is based on a divide-and-conquer strategy, exploiting subtle interactions between the bits of the Keccak internal state, and is of independent interest.

The low algebraic degree of a Keccak round has been exploited in many previous attacks, and in particular, in key recovery attacks [21], preimage attacks [6] and zero-sum distinguishers on the permutation [2, 11]. However, most of those attacks (apart from [21]) were only applied to the non-keyed Keccak variants, whereas we focus on its keyed modes. Furthermore, several of these attacks seem to have limited applicability, as they either assume a very powerful attack model (which does not correspond to a realistic attack scenario), or give a marginal improvement over generic attacks. Compared to these related attacks, our attacks seem to have broader applicability (as they focus on concrete schemes that use the Keccak permutation), and are significantly more efficient than generic attacks.

From a methodological point of view, most related attacks [2, 6, 11, 21] were based on algebraic analysis of the Keccak non-linear component. Although such analysis can be highly non-trivial (e.g., see [11]), it mostly takes into account the low algebraic degree of the Keccak permutation, but ignores several other (potentially useful) properties of Keccak internal mappings. The main difference between our approach and the previous ones, is that we show how to combine structural properties of Keccak (such as the limited diffusion of its linear layer) in order to gain advantage over standard algebraic analysis. In fact, it is likely that some of our techniques may be combined with the previous attacks to improve their results.

The paper is organized as follows. In Section 2, we briefly describe the cube attack, and in Section 3, we describe the Keccak sponge function and its keyed variants we analyse. In Section 4 we present our cube attacks on Keccak. In Section 5 we describe attacks which are based on output prediction (exploiting cube testers). Finally, we present our divide-and-conquer attack in Section 6 and conclude the paper in Section 7.

2 Cube Attacks

The cube attack is a chosen plaintext key-recovery attack, which was formally introduced in [14] as an extension of higher order differential cryptanalysis [20] and AIDA [23]. Since its introduction, the cube attack was applied to many different cryptographic primitives such as [3, 4, 21]. Below we give a brief description of the cube attack, and refer the reader to [14] for more details.

The cube attack assumes that the output bit of a cipher is given as a black-box polynomial $f : X^n \rightarrow \{0, 1\}$ in the input bits (variables). The main observation used in the attack is that when this polynomial has a (low) algebraic degree d , then summing its outputs over 2^{d-1} inputs, in which a subset of variables (i.e., a cube) of dimension $d - 1$ ranges over all possible values, and the other variables are fixed to some constant, yields a linear function (see the theorem below).

Theorem 1. (Dinur, Shamir) *Given a polynomial $f : X^n \rightarrow \{0, 1\}$ of degree d . Suppose that $0 < k < d$ and t is the monomial $x_0 \dots x_{k-1}$. Write the function as*

$$f(x) = t \cdot P_t(x) + Q_t(x)$$

where none of the terms in $Q_t(x)$ is divisible by t . Note that $\deg P_t \leq d - k$. Then the sum of f over all values of the cube (defined by t) is

$$\sum_{x'=(x_0, \dots, x_{k-1}) \in C_t} f(x', x) = P_t(\underbrace{1, \dots, 1}_k, x_k, \dots, x_{n-1})$$

whose degree is at most $d - k$ (or 1 if $k = d - 1$), where the cube C_t contains all binary vectors of the length k .

A simple combinatorial proof of this theorem is given in [14]. Algebraically, we note that addition and subtraction are the same operation over $GF(2)$. Consequently, the cube sum operation can be viewed as differentiating the polynomial with respect to the cube variables, and thus its degree is reduced accordingly.

2.1 Preprocessing (Offline) Phase

The preprocessing phase is carried out once per cryptosystem and is independent of the value of the secret key.

Let us denote public variables (variables controlled by the attacker e.g., a message or a nonce) by $v = (v_1, \dots, v_p)$ and secret key variables by $x = (x_1, \dots, x_n)$. An output (ciphertext bit, keystream bit, or a hash bit) is determined by the polynomial $f(v, x)$. We use the following notation

$$\sum_{v \in C_t} f(v, x) = L(x)$$

for some cube C_t , where $L(x)$ is called the *superpoly* of C_t . Assuming that the degree of $f(v, x)$ is d , then, according to the main observation, we can write

$$L(x) = a_1x_1 + \dots + a_nx_n + c.$$

In the preprocessing phase we find linear superpolys $L(x)$, which eventually help us build a set of linear equations in the secret variables. We interpolate the linear coefficients of $L(x)$ as follows

- find the constant $c = \sum_{v \in C_t} f(v, 0)$
- find $a_i = \sum_{v \in C_t} f(v, 0, \dots, \underbrace{1}_{x_i}, 0, \dots, 0)) = a_i$

Note that in the most general case, the full symbolic description of $f(v, x)$ is unknown and we need to estimate its degree d using an additional complex preprocessing step. This step is carried out by trying cubes of different dimensions, and testing their *superpolys* $L(x)$ for linearity. However, as described in our specific attacks on Keccak, the degree of $f(v, x)$ can be easily estimated in our attacks, and thus this extra step is not required.

2.2 Online Phase

The online phase is carried out after the secret key is set. In this phase, we exploit the ability of the attacker to choose values of the public variables v . For each cube C_t , the attacker computes the binary value b_t by summing over the cube C_t or in other words

$$\sum_{v \in C_t} f(v, x) = b_t.$$

For a given cube C_t , b_t is equal to the linear expression $L(x)$ determined in the preprocessing phase, therefore a single linear equation is obtained

$$a_1x_1 + \dots + a_nx_n + c = b_t.$$

Considering many different cubes C_t , the attacker aims at constructing a sufficient number of linear equations. If the number of (linearly independent) equations is equal to a number of secret variables, the system is solved by the Gaussian elimination.¹

2.3 Cube Testers

The notion of cube testers was introduced in [3], as an extension of the cube attack. Unlike standard cube attacks, cube testers aim at detecting a non-random behaviour (rather than performing key recovery), e.g., by observing that the cube sums are always equal to zero, regardless of the value of the secret key.

¹ More generally, one can use any number of linearly independent equations in order to speed up exhaustive search for the key.

3 Keccak Sponge Function

Keccak is a family of sponge functions [7]. It can be used as a hash function, but can also generate an infinite bit stream, making it suitable to work as a stream cipher or a pseudorandom bit generator. In this section, we provide a brief description of the Keccak sponge function to the extent necessary for understanding the attacks described in the paper. For a complete specification, we refer the interested reader to the original specification [8].

The sponge function works on a b -bit internal state, divided according to two main parameters r and c , which are called bitrate and capacity, respectively. Initially, the $(r+c)$ -bit state is filled with 0's, and the message is split into r -bit blocks. Then, the sponge function processes the message in two phases.

In the first phase (also called the absorbing phase), the r -bit message blocks are XORed into the state, interleaved with applications of the internal permutation. After all message blocks have been processed, the sponge function moves to the second phase (also called the squeezing phase). In this phase, the first r bits of the state are returned as part of the output, interleaved with applications of the internal permutation. The squeezing phase is finished after the desired length of the output digest has been produced.

Keccak is a family of sponge functions defined in [8]. The state of Keccak can be visualized as an array of 5×5 lanes, where each lane is a 64-bit string in the default version (and thus the default state size is 1600 bits). Other versions of Keccak are defined with smaller lanes, and thus smaller state sizes (e.g., a 400-bit state with a 16-bit lane). The state size also determines the number of rounds of the Keccak-f internal permutation, which is 24 for the default 1600-bit version.

All Keccak rounds are the same except for the round-dependent constants, which are XORed into the state. Below there is a pseudo-code of a single round. In the latter part of the paper, we often refer to the algorithm steps (denoted by Greek letters) described in the following pseudo-code.

```

Round(A,RC) {
    θ step
    C[x] = A[x,0] xor A[x,1] xor A[x,2] xor
           A[x,3] xor A[x,4],                                forall x in (0...4)
    D[x] = C[x-1] xor rot(C[x+1],1),                      forall x in (0...4)
    A[x,y] = A[x,y] xor D[x],                                forall (x,y) in (0...4,0...4)

    ρ step                                                 forall (x,y) in (0...4,0...4)
    A[x,y] = rot(A[x,y], r[x,y]),

    π step                                                 forall (x,y) in (0...4,0...4)
    B[y,2*x+3*y] = A[x,y],

    χ step                                                 forall (x,y) in (0...4,0...4)
    A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]),
}

```

```

 $\iota$  step
A[0,0] = A[0,0] xor RC

return A }

```

All the operations on the indices shown in the pseudo-code are done modulo 5. A denotes the complete permutation state array and $A[x,y]$ denotes a particular lane in that state. $B[x,y]$, $C[x]$, $D[x]$ are intermediate variables. The constants $r[x,y]$ are the rotation offsets, while RC are the round constants. $\text{rot}(W,m)$ is the usual bitwise rotation operation, moving bit at position i into position $i+m$ in the lane W ($i+m$ are done modulo the lane size). θ is a linear operation that provides diffusion to the state. ρ is a permutation that mixes bits of a lane using rotation and π permutes lanes. The only non-linear operation is χ , which can be viewed as a layer of 5-bit S-boxes. Note that the algebraic degree of χ over $GF(2)$ is only 2. Furthermore, χ only multiplies neighbouring bits ($A[x,y,z]$ and $A[x+1,y,z]$). Finally, ι XORs the round constant into the first lane.

In this paper we refer to the linear steps θ , ρ , π as the first half of a round, and the remaining steps χ and ι as the second half of a round. In many cases it is useful to treat the state as the 5×5 array of 64-bit lanes. Each element of the array is specified by two coordinates, as shown in Figure 1.

[0,0]	[1,0]	[2,0]	[3,0]	[4,0]
[0,1]	[1,1]	[2,1]	[3,1]	[4,1]
[0,2]	[1,2]	[2,2]	[3,2]	[4,2]
[0,3]	[1,3]	[2,3]	[3,3]	[4,3]
[0,4]	[1,4]	[2,4]	[3,4]	[4,4]

Fig. 1. Lanes coordinates. Each square represents a lane in the state.

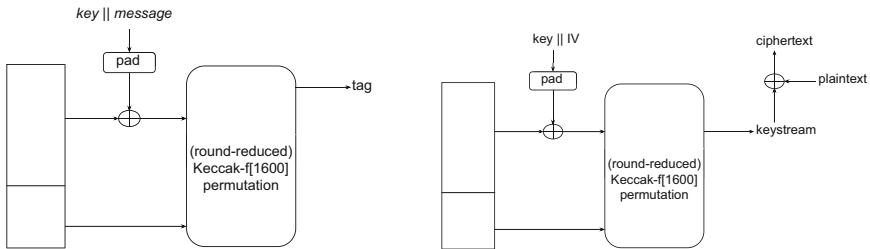
3.1 Keyed Modes of Keccak

The Keccak sponge function can be used in keyed mode, providing several different functionalities. Three of these functionalities which we analyse in this paper are a hash-based message authentication code (MAC), a stream cipher and an authenticated encryption (AE) scheme based on the design methods proposed in [10].

MAC Based on Keccak. A message authentication code (MAC) is used for verifying data integrity and authentication of a message. A secure MAC is expected to satisfy two main security properties. Assuming that an adversary has access to many valid message-tag pairs, (1) it should be infeasible to recover the secret key used and (2) it should be infeasible for the adversary to forge a MAC, namely, provide a valid message-tag pair (M, T) for a message M that has not been previously authenticated.

A hash-based algorithm for calculating a MAC involves a cryptographic hash function in combination with a secret key. A typical construction of such a MAC is HMAC, proposed by Bellare et al. [5]. However, for the Keccak hash function, the complex nested approach of HMAC is not needed and in order to provide a MAC functionality, we simply prepend the secret key to the message in order to calculate a tag. In this paper, we only use short messages such that the internal permutation is applied only once (see Figure 2).

Stream Cipher Based on Keccak. In the stream cipher mode, the state is initialized with the secret key, concatenated with a public initialization vector (IV). After each Keccak permutation call, an r -bit keystream (where r is the bitrate of the Keccak instance) is extracted and used to encrypt a plaintext via bitwise XOR. In this paper, we only exploit the first r bits of keystream, such that the internal permutation is applied only once (as shown in Figure 3).



Authenticated Encryption Scheme Based on Keccak. The Keccak sponge function can also be used as a building block for authenticated encryption (AE) schemes, simultaneously providing confidentiality, integrity, and authenticity of data. In this paper, we concentrate on the concrete design of Keyak [9] — a recently proposed authenticated encryption scheme, submitted to the CAESAR competition [1]. The scheme is based on the Keccak permutation with a nominal number of rounds set to 12.

Figure 4 shows the scheme of Lake Keyak, which is the primary recommendation of the Keyak family algorithms. In this scheme, the key and tag sizes are 128 bits long, and the capacity is set to $c = 252$ (i.e., $r = 1600 - 252 = 1348$). For a more formal description, we refer the reader to [9].

Figure 4 shows how the Keyak scheme processes two plaintext blocks. The first permutation call of Keyak takes as an input a key and a nonce. We note that some of our attacks use up to 2 plaintext blocks, and further note that the scheme has an optional input of associated data, which we do not use.

According to the specification of Keyak, in order to assure confidentiality of data, a user must respect the nonce requirement. Namely, a nonce cannot be reused, otherwise, confidentiality is not guaranteed. However, for authenticity and integrity of data, a variable nonce is not required (according to the Keyak specification).

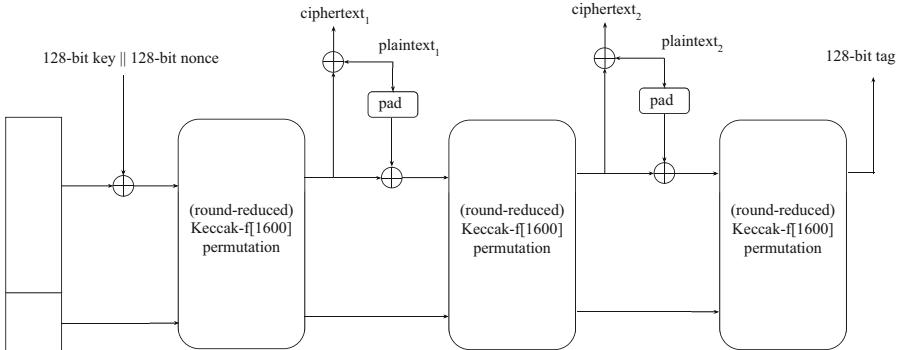


Fig. 4. Lake Keyak processing two plaintext blocks

3.2 Attack Models and Parameters

Most of our attacks focus on default variants of Keccak, suggested by the Keccak designers, namely 1600-bit state and 1024-bit bitrate, or a 1344-bit bitrate (used for example in Keyak and SHAKE-128). Furthermore, we concentrate on typical, real-life key, tag and IV lengths, avoiding artificial scenarios although they could potentially help the attacker (e.g., tags larger than 256 bits or very long IVs).

The Keccak sponge function can also work on smaller states, which may be useful for lightweight cryptography. In such Keccak variants, the size of the internal lanes is reduced and this has an effect on our attacks, as we highlight in several places.

All the attacks follow the chosen plaintext model, assuming that the attacker is able to choose various values of message/nonce/IV and obtain the corresponding ciphertext/tag/keystream outputs. For Keccak working as a MAC, we can control many input message bits, but only a short tag (128 or 256 bits) is available as an output. In the stream cipher mode the situation is reversed, as the attacker can only control IV bits (typically up to 256 bits), however, the output (keystream bits) can be as big as the bitrate (1024 bits for the default variant). Interestingly, for authenticated encryption mode (such as Keyak), we can take advantage of both long input and available output, as shown in Section 6. For some of our attacks in Sections 5 and 6 we assume a stronger model of attacks, where a nonce is reused.

4 Cube Attack on Keccak Sponge Function

In this section we focus on practical key recovery attacks which can be implemented and verified on a desktop PC. We analyse the round-reduced Keccak used as a MAC and a stream cipher. We also show how to recover the key for the 6-round Keyak.

4.1 Key Recovery Attack on 5-Round Keccak Working as MAC

We attack the default variant of Keccak with a 1600-bit state ($r = 1024$, $c = 576$), where the number of rounds of the internal permutation is reduced to 5. The key and tag sizes are both 128 bits.

Preprocessing Phase. As previously noted, we exploit the property that the algebraic degree of a single round of the Keccak permutation is only 2. Therefore, after 5 rounds the algebraic degree is at most $2^5 = 32$ and for any cube with 31 variables, the superpoly consists of linear terms only. In general, cubes of dimension 30 (or smaller) are not expected having linear superpolys. Although this may occur by chance, considering such cubes somewhat complicates preprocessing, as we need to perform linearity tests. For some cubes it is expected that superpolys will be constant, which are not useful for key-recovery attacks (as they do not contain information about the key). This typically occurs due to the slow diffusion of variables into the initial rounds, which causes the algebraic degree of the examined output bits to be less than the maximal possible degree of 32.

To find useful cubes for our attack, we randomly pick 31 out of the 128 public variables and check whether the superpoly consists of any secret variables or it is a constant. With this simple strategy, we have been able to find 117 linearly independent expressions (superpolys) in a few days on a desktop PC (example is given in Appendix A). The search was more complex than expected, as only 20 – 25% of the superpolys are useful (i.e., non-constant). On the other hand, we found more superpolys and shortened the search time by examining different output bits (with their corresponding superpolys).

Online Phase

In the online phase, the attacker computes the actual binary value of a given superpoly by summing over the outputs obtained from the corresponding cube. There are 19 cubes used in this attack, each cube with 31 variables. Thus, the attacker obtains $19 \cdot 2^{31} \cong 2^{35}$ outputs for the 5-round Keccak. Having computed the values of the superpolys, the attacker constructs a set of 117 linearly independent equations, and recovers the full 128-bit secret key by guessing the values of 11 additional linearly independent equations. In total, the complexity of the online phase is dominated by 2^{35} Keccak calls (the cost of linear algebra can be neglected).

4.2 Key Recovery Attack on 6-round Keccak Working in Stream Cipher Mode

A direct extension of the attack to 6 rounds seems infeasible as we would deal with polynomials of approximate degree $2^6 = 64$, and it is very unlikely to find (in reasonable time) cubes with linear superpolys. However, one more round can

be reached by exploiting a specific property of the Keccak χ step. As χ operates on the rows independently, if a whole row (5 bits) is known, we can invert these bits through ι and χ from the given output bits. Consequently, the final nonlinear step χ can be inverted and the cube attack is reduced to 5.5 rounds. As the first half of a round is linear and does not increase the degree, the output bits have a manageable polynomial degree of at most 32 and the scenario is very similar to the one considered in the previous attack.

Standard MACs are of size 128 or 256 output bits, which are insufficient for inversion — these output bits do not allow us to uniquely calculate any bit (or a linear combination of bits) after 5.5 rounds. If we consider longer MACs (for instance with 320 bits), then the attack setting becomes somewhat artificial. However, we can still attack the Keccak sponge function working in a different mode, where the attacker has access to more output bits. A good example is Keccak used as a stream cipher, and here, we attack the default variant of Keccak with 1600-bit state, $r = 1024$, $c = 576$ with key and IV sizes of 128 bits. The first 960 of the 1024 available output bits contain $960/5 = 192$ full rows (each sequence of 320 bits contains $320/5 = 64$ full rows), which can be inverted and exploited in the attack.

We executed the preprocessing phase in a similar way to the one described for the 5-round attack. We were able to find 128 linearly independent superpolys using 25 cubes (example is given in Appendix B). This gives an online attack complexity of $2^{31} \cdot 25 \cong 2^{36}$.

4.3 Key Recovery Attack on MAC-Based State-Reduced 6-Round Keccak

We attack the Keccak MAC that operates on 400-bit state with 80-bit key and 160-bit bitrate. As the state is smaller, 128 bits of MAC (output bits) cover all the rows in the state and we are able to invert these rows through the ι and χ steps. Therefore, our attack on the 6-round Keccak MAC becomes practical.

During the preprocessing phase, we have found 80 linearly independent superpolys using 18 cubes. This allows us to recover the 80-bit secret key with complexity $2^{31} \cdot 18 \cong 2^{35}$. It is interesting to note that, compared to the previous attacks, the superpolys consist of many more secret variables. It is due to a faster diffusion of variables when the state is smaller. An example of a cube chosen for the attack is given in Appendix C.

4.4 Key Recovery Attack on 6-Round Keyak

The key recovery attack is essentially the same as the one described for the stream cipher mode. Instead of IV variables, we use the nonce as cube variables. After a single call to the Keccak permutation, the bitrate part of state is available (by XORing known plaintext with the corresponding ciphertext – see Figure 4). As in the stream cipher mode, we have many output bits available ($r = 1348$), allowing to easily invert ι and χ and break 6 rounds.

5 Output Prediction for Keyed Variants of Keccak

In this section, we first present a practical cube tester for 6.5-round Keccak, and then show how to exploit similar distinguishers in order to predict the output of Keccak when used as a MAC (i.e., mount a forgery attack) or in stream cipher mode.

5.1 Practical Cube Tester for 6.5-Round Keccak Permutation

We show how to construct a practical cube tester for the 6.5-round Keccak permutation. As the expected algebraic degree for 6-round Keccak is 64, such an attack may seem at first impractical (without exploiting some internal invertibility properties, as in the previous section). However, if we carefully choose the cube variables, we can exploit a special property of θ in order to considerably reduce the output degree after 6 rounds and keep the complexity low.

The well-known property of θ , we exploit, is that its action depends on the column parities only (and not on the actual values of each bit in a column). Thus, if we set the cube variables in such a way that all the column parities are constants for all the possible variable values, then θ will not diffuse these variables throughout the state. Moreover, as ρ and π only permute the bits of the state, it is easy to choose the cube variables such that after the linear part of the round, they are not multiplied with each other through the subsequent non-linear χ layer. Consequently, the algebraic degree of the state bits in the cube variables remains 1 after the first round, and it is at most 32 after 6 rounds.

We choose the 33-dimensional cube $\{v_0, v_1, \dots, v_{32}\}$ such that $v_i = A[0, 2, i]$, while ensuring that the column parities remain constant by setting the additional linear constraints $A[0, 3, i] = v_i \oplus c_i$, for arbitrary binary constants c_i (see Figure 5). In other words, we sum over the outputs of the 33-dimensional linear subspace defined on the 66 state bits $A[0, 2, i], A[0, 3, i]$ by 33 equations $A[0, 2, i] = A[0, 3, i] \oplus c_i$ for $i \in \{0, 1, \dots, 32\}$. The remaining bits of the input state (some of which are potentially unknown secret variables) are set to arbitrary constants. As can be seen from Figure 5, at the input to χ , each 5-bit row of the state contains at most one variable, and therefore, the variables are not multiplied together in the first round as required.

Since the degree of the output polynomials in the cube variables after 6 rounds is only 32, the cube sum of any output bit after 6 rounds is equal to zero, which is a clear non-random property. Moreover, we can add a (linear) half-round and obtain a 6.5-round distinguisher using the same cube. Furthermore, if we assume that we can obtain sufficiently many output bits in order to partially invert the non-linear layer (as in the previous section), we can extend the attack to 7 rounds in practical time. Note that the distinguishing attack works regardless of the number of secret variables, their location, or their values.

Assume that the 33-dimensional cube is missing an output for one value of the public variables. Then, as the sums of all 2^{33} outputs is zero, the missing output is equal to the sum of the remaining $2^{33} - 1$ outputs. Thus, the distinguishing attack can be used to predict the output of the cipher for a previously unseen

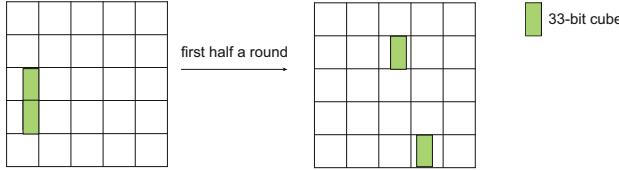


Fig. 5. The initial state of a cube tester and the transition through the first linear part of the round (θ , ρ , π steps)

input value. In the rest of this section, we exploit this property in more specific attacks on keyed modes of Keccak.

We note that it is possible to predict the values of several outputs at a smaller amortized cost than summing over a 33-variable cube for each missing value. This can be accomplished by using larger cubes which contain several missing outputs, and using more complex algebraic algorithms (such as the ones used in [6]) in order to calculate them. However, in this paper we focus on basic attacks that predict only one output.

5.2 Extending the Cube Tester to Smaller States and More Rounds

When considering Keccak variants with states of at most $b = 400$ bits, then each lane contains (at most) 16 bits, and it is not clear how to select the 33-variable cube as in the previous case. However, we can generalize the above idea by noticing that it is possible to carefully select a cube of dimension of (up to) $4 \cdot 16 = 64$ such that its variables are not multiplied together in the first round. Such a cube contains (up to) 64 independent variables in 4 lanes of 16 columns, where the 5th lane keeps the column parities constant (e.g., to keep the column parities to zero, its value has to be equal to the XOR of the 4 independent lanes). One can observe that after the application of ρ and π , these variables are not multiplied together by χ in the first round.

A careful selection of cube variables allows to select large cubes for which the complexity of the distinguishing attack is significantly reduced compared to the complexity with arbitrary cubes. We note that there exist other methods to carefully select a large set of variables that are not multiplied together in the first round, and we only give one of these methods in this section.

Clearly, the idea can also be exploited for Keccak variants with larger states, for which we can select larger cubes and mount distinguishing attacks on more than 6.5 rounds. In such cases, the attack becomes impractical, but it may still be more efficient than generic attacks (depending on the attack setting).

5.3 MAC Forgery for 7-Round Keccak MAC and Keyak

We attack 7-round Keccak MAC with the default bitrate $r = 1024$ and 128-bit key and tag length, ideally providing 128-bit security. In order to forge a tag T for an arbitrary message M , we use a cube tester similar to the one from

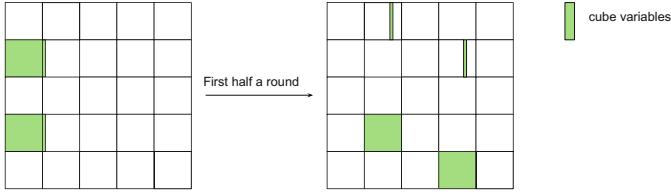


Fig. 6. The transition through the first half a round for the cube tester exploited in the MAC forgery attack. In the initial state cube variables are assumed to be equal column-wise.

Section 5.1. We choose a 65-variable cube, as shown in Figure 6. The remaining bits of the input state are set according to the message bits of M which are not part of the cube and additional constants (some of which are potentially unknown secret variables). Due to the placement of the cube variables, we can go through the first round without increasing the algebraic degree. Consequently, after 7 rounds, the degree is at most 64 (rather than 128). Therefore, the cube sum of any output bit after 7 rounds is equal to zero.

The forgery attack works by collecting $2^{65} - 1$ tags for chosen messages which consist of all 2^{65} messages defined by the cube (shown in Figure 6), with the exception of M . Since the cube sums of all 128 output bits of the MAC are zero, we can calculate the tag of M by XORing the $2^{65} - 1$ known tags. Therefore, we forge a valid message-tag pair (M, T) , which has not been previously seen.

MAC Forgery for 7-round Keyak. For Keyak, the selection of cube variables is limited to the 128 nonce bits, and it is not clear how to exploit the method above to gain an extra round. However, according to the Keyak specification, a variable nonce is not required for authenticity and integrity of data. Therefore, we fix the nonce, and hence also fix the state after the first permutation call. Then, before the second permutation call, the state absorbs plaintext variables, which we select as cube variables. In this setting, we can control as many as $r = 1348$ state bits, and forge the 128-bit tag with complexity 2^{65} , similarly to the 7-round Keccak MAC forgery above.

5.4 MAC Forgery for 8 Rounds

We attack 8-round variants with a longer 256-bit key and tag, increasing the security level to 256 bits. The other parameters remain the same. In this case, we select a 129-variable cube as follows: 128 variables among lanes $A[4, 0], A[4, 1], A[4, 2]$ (as $c = 576$, these lanes contain public message bits), using the generalized idea of Section 5.2, and 1 additional variable in lanes $A[2, 1], A[2, 2]$. After the linear layer, these variables diffuse to lanes $A[2, 0], A[1, 1], A[1, 2], A[0, 3], A[2, 4]$ which have different y indices and are not multiplied together in the first round. Therefore, for such a selection, the output degree in the variables after 8 rounds is at most $2^7 = 128$, implying that the cube sums are zero, and we can forge a message with complexity 2^{129} .

5.5 Keystream Prediction for 8- and 9-Round Keccak-Based Stream Cipher

The output prediction strategy used to forge a MAC can be used to predict the keystream of a previously unseen IV in the stream cipher mode. The difference is that in this mode we generally have less control over the public variables (IV), and we cannot select the cube variable as in the previous attacks to gain a round at the beginning. On the other hand, we exploit larger cubes than in MACs (as some stream ciphers aim for higher security level compared to MACs). Furthermore, as more output bits are generally available, we can invert the last non-linear χ on sequences of 320 output bits and reduce their algebraic degree as in Section 4.

First, let us describe the attack on the 8-round variant with the default parameters $r = 1024$ and $c = 576$. We set the key length to 256 bits and IV length to 128 bits. Having 1024 bits of keystream, we can invert as many as 960 bits through ι and χ . Therefore, we reduce our attack to 7.5 rounds for which the algebraic degree is at most $2^7 = 128$. The attack has two phases.

Preprocessing Phase

Since for 7.5 rounds we deal with the algebraic degree 128, summing over any 128-bit cube gives a constant regardless of the secret key values. Thus, we determine a cube sum (either 1 or 0) for each output bit of the first $3 \cdot 320 = 960$ bits of state (which can be fully inverted online, given the 1024-bit keystream after 8 rounds). The cost of the preprocessing is 2^{128} Keccak calls.

Online Phase

Keystream prediction for an unused IV follows the same pattern as for a MAC forgery. First, we collect $2^{128} - 1$ keystream sets for IVs which reside in a 128-dimensional cube (not including the IV whose keystream we predict), and invert ι and χ on the first available bits $3 \cdot 320 = 960$ bits of state. Then, the first 960 bits of keystream of the remaining IV can be predicted. This is done by XORing the 960 bits of all the $2^{128} - 1$ inverted keystreams to the cube sums calculated during preprocessing, and then reapplying χ and ι to the outcome. The complexity of the online phase is 2^{128} Keccak calls, whereas the generic attack complexity is 2^{256} .

Keystream Prediction for 9-Round Keccak

Following the procedure from 8-round variant, we can easily extend the attack to 9-round Keccak with $r = 1024, c = 576$ using larger keys of 512 bits, and an IV of 256 bits. The only difference is that we would sum over larger 256-bit cubes. According to the recent analysis [18], the ideal security level of this variant should be 512 bits², whereas our attack has complexity of 2^{256} .

² According to [18], the security of the scheme is $\min(n, c, b/2) = \min(512, 576, 800) = 512$ bits.

6 Divide-and-Conquer Key Recovery Attack on Keccak-Based MAC and Keyak

In the previous section, we showed how to predict output for several keyed variants of Keccak. In this section, we return to the most powerful type of attacks, and describe key recovery attacks on the 6- and 7-round Keccak with a 1600-bit state. We attack a variant with the capacity parameter $c = 256$ and a 128-bit key. Therefore, the security level of this variant is 128 bits.

First, we note that for 6 rounds the degree of the output bits is generally $2^6 = 64$, and thus the preprocessing phase of the standard cube attack is too expensive to perform (without exploiting some internal invertibility properties, as in Section 4). Another possible approach is to carefully select the cube such that we obtain a practical distinguisher (as in Section 5.1). Then, we can try to apply several techniques that were developed to exploit similar distinguishers for key recovery (such as conditional differential cryptanalysis [19] and dynamic cube attacks [15]). However, these techniques seem to be better suited for stream ciphers built using feedback shift registers, rather than the SP-network design of Keccak.

As it is not clear how to use the standard key recovery techniques in our case, we use a different approach. The main idea in our attack is to select the public variables of the cube in such a way that the superpolys depend only on a (relatively) small number of key bits, whose value can be recovered independently of the rest of the key. Thus, the full key can be recovered in several phases in a divide-and-conquer manner.

The approach we use is similar to the one used in the attacks on the stream ciphers Trivium and Grain in [16]. However, while the results on Trivium and Grain were mostly obtained using simulations, our attack is based on theoretical analysis that combines algebraic and structural properties of Keccak in a novel way. This analysis enables us to estimate the complexity of the attack beyond the feasible region (in contrast to the simulation-based attack of [16]).

Borderline Cubes. The starting point of the attack is the cube tester of Section 5.1, which is based on a 33-variable cube, whose column parities remain constant for all of their 2^{33} possible values. As the cube variables are not multiplied together in the first round and the degree of 6-round Keccak in the state variables after one round is $2^5 = 32$, then the cube sums for the 33-variables are zero for all output bits. When we remove one variable from this cube, the sums are no longer guaranteed to be zero and they depend on the values of some of the constant bits of the state. This leaves us in a borderline situation. If a state bit is not multiplied with the cube variables in the first round, then the cube sums do not depend on the value of this bit. On the other hand, if a state bit is multiplied with the cube variables in the first round, then the cube sums generally depend on the value of the bit (assuming sufficient mixing of the state by the Keccak mapping). Thus, by a careful selection of a “borderline” cube of dimension 32, we can assure that the cube sums depend only on a (relatively)

small number of key bits. This gives rise to divide-and-conquer attacks, which we describe in detail in the rest of this section.

6.1 Basic 6-Round Attack

According to the Keccak MAC specification, the 128-bit key is placed in $A[0, 0]$ and $A[1, 0]$. However, it is worth noting that our attack could be easily adapted to any other placements of the secret key. We select 32 cube variables v_1, v_2, \dots, v_{32} in $A[2, 2]$ and $A[2, 3]$, such that the column parities of $A[2, *]$ remain constant for the 2^{32} possible values of the variables (similarly to Section 5.1). This careful selection of the cube variables leads to two properties on which our attack is based:

Property 1. The cube sum of each output bit after 6 rounds does not depend on the value of $A[1, 0]$.

Property 2. The cube sums of the output bits after 6 rounds depend on the value of $A[0, 0]$.

The detailed proof of these properties is given in Appendix D, but note that as we selected a “borderline” cube of 32 variables, we can prove Property 1 by showing that the cube variables are not multiplied with the secret variables of $A[1, 0]$ in the first round. Similarly, we can prove Property 2 by showing that the cube variables are multiplied with the secret variables of $A[0, 0]$ in the first round.

We now describe the attack which exploits the two properties to retrieve the value of $A[0, 0]$. For the sake of convenience, we separate the attack to preprocessing and online phases, where the preprocessing phase does not depend on the online values of the secret key. However, we take into account both of the phases when calculating the complexity of the full attack. The preprocessing phase is described below.

1. Set the capacity lanes ($A[1, 4], A[2, 4], A[3, 4], A[4, 4]$) to zero. Set all other state bits (beside $A[0, 0]$ and the cube variables) to an arbitrary constant.³
2. For each of the 2^{64} possible values of $A[0, 0]$:
 - (a) Calculate the cube sums after 6 rounds for all the output bits. Store the cube sums in a sorted list L , next to the value of the corresponding $A[0, 0]$.

As the cube contains 32 variables, the time complexity of Step 2.(a) is 2^{32} . The cube sums are calculated and stored for each of the 2^{64} values of $A[0, 0]$, and thus the total time complexity of the preprocessing phase is $2^{64} \cdot 2^{32} = 2^{96}$, while its memory complexity is 2^{64} .

The online phase, which retrieves $A[0, 0]$, is described below.

³ The chosen constant has to include padding bits.

1. Request the outputs for the 2^{32} messages that make up the chosen cube (using the same constant as in the preprocessing phase).
2. Calculate the cube sums for the output bits and search them in L .
3. For each match in L , retrieve $A[0, 0]$ and store all of its possible values.

Although the actual online value of $A[1, 0]$ does not necessarily match its value used during preprocessing, according to Property 1, it does not affect the cube sums. Thus, we will obtain a match in Step 3 with the correct value of $A[0, 0]$. In order to recover $A[1, 0]$, we independently apply a similar attack using 32 public variables in $A[4, 2]$ and $A[4, 3]$ (for which properties corresponding to Property 1 and Property 2 would apply). Finally, in order to recover the full key, we enumerate and test all combinations of the suggestions independently obtained for $A[0, 0]$ and $A[1, 0]$.

The time complexity of the attack depends on the number of matches we obtain in Step 3. The expected number of matches is determined by several factors, and in particular, it depends on a stronger version of Property 2, namely on the actual distribution of the cube sums after 6 rounds in $A[0, 0]$ (Property 2 simply tells us that the distribution is not concentrated in one value of $A[0, 0]$). Furthermore, the number of matches varies according to the number of available output bits, and the actual cube and constants chosen during preprocessing Step 1 (and reused online). In general, assuming that the cube sums are uniformly distributed in $A[0, 0]$, and we have at least 64 available output bits (which is the typical case for a MAC), we do not expect more than a few suggestions for the 64-bit $A[0, 0]$ in online Step 3. Although we cannot make the very strong assumption that the cube sums are uniformly distributed in $A[0, 0]$, our experiments (described in Appendix D) indeed reveal that we are likely to remain with very few suggestions for $A[0, 0]$ in online Step 3. Furthermore, even if we remain with more suggestions than expected, we can collect sums from several cubes, obtained by choosing different cube variables or changing the value of the message bits, which do not depend on the cube variables. This reduces the number of matches in Step 3 at the expense of slightly increasing the complexity of the attack. We thus assume that the number of matches we obtain in Step 3 is very small.

The online phase requires 2^{32} data to retrieve the 64-bit $A[0, 0]$ and requires 2^{32} time in order to calculate the cube sums. As previously mentioned, in order to recover $A[1, 0]$, we independently apply the same attack but this time using 32 public variables in $A[4, 2]$ and $A[4, 3]$. Thus, for the full key recovery, the total data complexity is 2^{33} and the online time complexity is 2^{33} (assuming that we do not have too many suggestions in Step 3). Taking preprocessing into account, the total time complexity is 2^{96} , and the memory complexity is 2^{64} .

6.2 Balanced 6-Round Attack

The basic attack above employs an expensive preprocessing phase which dominates its time complexity. In this section, we describe how to tradeoff the complexity of the preprocessing and online phases, allowing us to devise a more efficient attack.

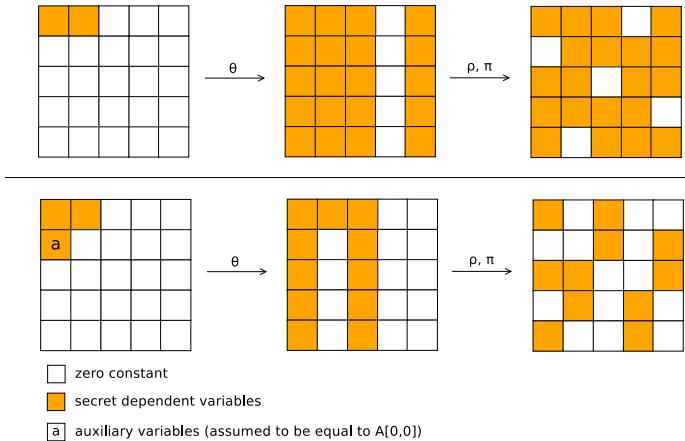


Fig. 7. Impact of auxiliary variables on diffusion of the secret key variables. When using auxiliary variables, $A[0, 0]$ (secret variables) and $A[0, 1]$ (auxiliary variables) are diffused to $A[0, 0]$ and $A[1, 3]$, without affecting many lanes of the state.

The imbalance of the basic attack comes from the fact that the cube sums after 6 rounds depend on all the variables of $A[0, 0]$. Thus, we need to iterate over all of their 2^{64} values during preprocessing to cover all the possible values of the cube sums in the online phase.

In order to reduce the preprocessing complexity, we aim to eliminate the dependency of the cube sums on some of the variables of $A[0, 0]$. However, it is not clear how to achieve this, as we cannot control the values of the secret variables and thus we cannot directly control their diffusion. On the other hand, as the action of θ is only determined by the column parities, we can indirectly control the diffusion of the secret variables by using additional *auxiliary variables* in the lanes with $x = 0$, and specifically in $A[0, 1]$. If we set the column parities for $x = 0$ to zero (or any other pre-defined constant), then the diffusion of the secret key is substantially reduced. Figure 7 shows an impact of the auxiliary variables on diffusion of the secret key variables.

Similarly to the basic attack, we select a borderline cube with 32 variables in $A[2, 2]$ and $A[2, 3]$, such that the column parities of $A[2, *]$ remain constant for the 2^{32} possible values of the variables. As explicitly shown in the proof of Property 1 in Appendix D, the cube variables are not multiplied with the auxiliary variables or secret variables in the first round (assuming that the column parities of $x = 0$ are fixed). Therefore, the cube sums after 6 rounds depend neither on the value of $A[0, 0]$, nor on the auxiliary variables of $A[0, 1]$ (but only on the column parities of $x = 0$). This observation gives rise to our balanced attack. Similarly to the basic attack, we divide the attack into preprocessing and online phases, where the preprocessing phase is described below.

1. Set the state bits (which are not cube variables) to zero (or an arbitrary constant). Furthermore, set $A[1, 0]$ and the 32 LSBs of $A[0, 0]$ to zero (or an arbitrary constant).
2. For each possible value of the 32 MSBs of $A[0, 0]$:
 - (a) Calculate the cube sums after 6 rounds for all the output bits. Store the cube sums in a sorted list L , next to the value of the 32 MSBs of $A[0, 0]$.

The cube sums are calculated and stored for each of the 2^{32} values of the 32 MSBs of $A[0, 0]$, and thus the total time complexity of the preprocessing phase is $2^{32} \cdot 2^{32} = 2^{64}$, while its memory complexity is 2^{32} .

The online phase is described below.

1. For each possible value of the 32 LSBs of $A[0, 1]$:
 - (a) Request the outputs for the 2^{32} messages that make up the chosen cube with the 32 LSBs of $A[0, 1]$ set according to Step 1 (setting the same constant values in the state as in the preprocessing).
 - (b) Calculate the cube sums for the output bits and search them in L .
 - (c) For each match in L , retrieve the 32 MSBs of $A[0, 0]$. Assume that the 32 LSBs of $A[0, 0]$ are equal to the 32 LSBs of $A[0, 1]$ (the 32 column parities should be zero, as in the preprocessing phase). Then, given the full 64-bit $A[0, 0]$, exhaustively search $A[1, 0]$ using trial encryptions, and if a trial encryption succeeds, return the full key $A[0, 0], A[1, 0]$.

Once the value of the 32 LSBs of $A[0, 1]$ in Step 1 is equal to the 32 LSBs of the (unknown) $A[0, 0]$, the corresponding column parities are zero, and thus they match the column parities assumed during preprocessing. The actual values of the 32 LSBs of $A[0, 1]$ and $A[0, 0]$ (and the actual value of $A[1, 0]$) do not necessarily match their values during preprocessing. However, they do not influence the cube sums, and thus the attack recovers the correct key once the value of the 32 LSBs of $A[1, 0]$ is equal to the 32 LSBs of $A[0, 0]$.

The online phase requires $2^{32+32} = 2^{64}$ chosen messages to retrieve the 64-bit $A[0, 0]$. Assuming that we do not have too many suggestions in Step 3 (as assumed in the basic attack), it requires 2^{64} time in order to obtain the data and calculate the cube sums, and additional 2^{64} time to exhaustively search $A[1, 0]$ in Step 1(c). Taking preprocessing into account, the total time complexity of the attack is about 2^{66} , and its memory complexity is 2^{32} .

We note that it is possible to obtain additional tradeoffs between the preprocessing and online complexities by adjusting the number of auxiliary variables. However, in this paper we describe the attack with the best total time complexity only.

6.3 7-Round Attack

For a MAC based on the 7-round Keccak, the algebraic degree of the output in the state variables of round 1 is $2^6 = 64$. We can extend our 6-round attack to 7 rounds by selecting a borderline cube of 64-variables (i.e., a full lane) in

$A[2, 2]$ and $A[2, 3]$. As the cube consists of 32 more variables than the cube of the 6-round attack, the data complexity increases by a factor of 2^{32} , and the time complexity of both the preprocessing and online phases increases by the same factor (except for the exhaustive search for $A[1, 0]$ in online Step 1(c), which still requires 2^{64} time). Thus, the data complexity of the full 7-round attack is 2^{64} , its time complexity is 2^{97} , and its memory complexity remains 2^{32} .

Comparison with Standard Cube Attacks. One may claim that the 6-round attacks presented in this section are somewhat less interesting, as it seems reasonable that the standard cube attack (such as the ones presented in Section 4) would break the scheme in a similar time complexity of (a bit more than) $2^{2^6} = 2^{64}$. However, in order to mount the standard cube attack, we need to run a lengthy preprocessing phase whose outcome is undetermined, whereas our divide-and-conquer algorithm is much better defined. Furthermore, the divide-and-conquer attacks allow a wider range of parameters and can work with much less than 2^{64} data.

Despite its advantage in attacking 6 rounds, the real power of the divide-and-conquer attack introduced in this paper is demonstrated by the 7-round attack. Indeed, the standard cube attack on the 7-round scheme is expected to require more than $2^{2^7} = 2^{128}$ time, and is therefore slower than exhaustive search, whereas our divide-and-conquer attack breaks the scheme with complexity 2^{97} .

6.4 Application to 7-Round Keyak

We now apply the divide-and-conquer attack to 7-round Keyak. As in the forgery attack on 7-round Keyak of Section 5.3, we reuse the nonce and consider the message bits as public variables in order to have more freedom and gain an additional round at the beginning. Therefore, we only aim to break the authenticity and integrity of Keyak. Compared to the attack of Section 5.3 which allows to forge a single tag, the 7-round attack described here is significantly stronger. This attack recovers the secret key, after which the security of the system is completely compromised (e.g. one can immediately forge the tag of any message).

In the initial setting, all the 1600 state bits obtained after the first permutation are unknown, and we aim to recover them. Once this state is recovered, we can run the permutation backwards and recover the secret key. In order to recover the secret 1600-bit state, we first obtain the encryption of an arbitrary 2-block message whose first-block ciphertext reveals the value of $r = 1348$ bits of secret state. Then, during the actual attack, we choose messages that set these $r = 1348$ known bits to an arbitrary pre-fixed constant (e.g., zero), whose value is defined and used during the preprocessing phase of the attack. Using this simple idea, the number of secret state variables for Keyak is reduced from 1600 to $c = 252$ variables in $A[1, 4], A[2, 4], A[3, 4], A[4, 4]$. Note that although the key size is only 128 bits, we have a larger number of 252 secret variables.

In this attack, we use a borderline cube containing $d = 32$ variables. Recall that in the case of MAC-based Keccak, a 32-variable cube was used to attack 6 rounds, but here, we have a larger output of 1348 bits which allows to exploit

the inversion property from Section 4. Therefore, we can attack 7 rounds using a 32-bit borderline cube, exploiting the inversion property on $320 \cdot 4 = 1280$ output bits.

In the attack on the 6-round Keccak MAC, we selected the 32 cube variables by varying 64 bits in $A[2, 2]$ and $A[2, 3]$, which diffuse to different 64 bits after the linear layer. As χ only multiplies consecutive bits in a row, then each such bit is multiplied with 2 neighbouring bits in its row, and therefore the 64 bits are multiplied with 128 bits that remain constant during the cube summation. As we selected a borderline cube, these 128 constant bits are the only ones that effect the value of the cube summations (which is the crucial property on which the divide-and-conquer attack is based), and we refer to these bits here as *effective bits*. Some of the values of the 128 effective bits are unknown as they depend on linear combinations of secret variables (such a combination can either be a singleton bit, or a linear combination of several secret bits), which we refer to here as *effective secret expressions*. Note that since each effective bit contains at most one effective secret expression, then the number of effective secret expressions is upper bounded by the number of effective bits.

In the case of the 6-round attack on the Keccak MAC, only 64 of the 128 effective bits actually depend on secret material (i.e., the number of effective secret expressions is 64). In order to recover the 64 bits of effective secret expressions, the idea was to enumerate their values during preprocessing, store their cube sums, and compare these sums to the ones obtained online. Therefore, the complexity of the basic (non-balanced) attack was about $2^{64+32} = 2^{96}$.

In the case of 7-round Keyak, we have as many as 252 secret variables, which extensively diffuse over the state. Therefore, a selection of a cube similar to the 6-round attack on MAC will cause the 64 cube variables to be multiplied with (the maximal number of) 128 effective secret expressions (instead of 64) in the first round, increasing the complexity of the basic attack to about $2^{128+32} = 2^{160}$, much above the exhaustive search of 2^{128} . In order to reduce the number of effective secret expressions, we use the idea from Section 5.2, and choose the 32 cube variables among the 5 lanes with $x = 0$. More precisely, we set the 8 LSBs of the first 4 lanes $A[0, 0], A[0, 1], A[0, 2], A[0, 3]$ as independent cube variables (i.e., we have a total of $4 \cdot 8 = 32$ independent variables), while the 8 LSBs of $A[0, 4]$ act as “parity checks”. Using that selection of cube variables, we have only 40 bits (instead of 64) that depend on the cube variables. The first linear layer diffuses these 40 bits to $A[0, 0], A[2, 1], A[4, 2], A[1, 3], A[3, 4]$. These lanes have distinct y coordinates, and are therefore not multiplied together by χ — the condition to get the first round for ‘free’.

Once again, as χ multiplies each bit with two neighbouring bits in its row, the 40 bits that depend on the cube variables are multiplied with $40 \cdot 2 = 80$ effective bits, which implies that the number of effective secret expressions is at most 80. Figure 8 shows an example diffusion of the 40-bit cube and the placement of the effective bits.

We now use the same procedure that we used for the basic attack on 6-round MAC to recover the values of the 80 effective secret expressions. Namely,

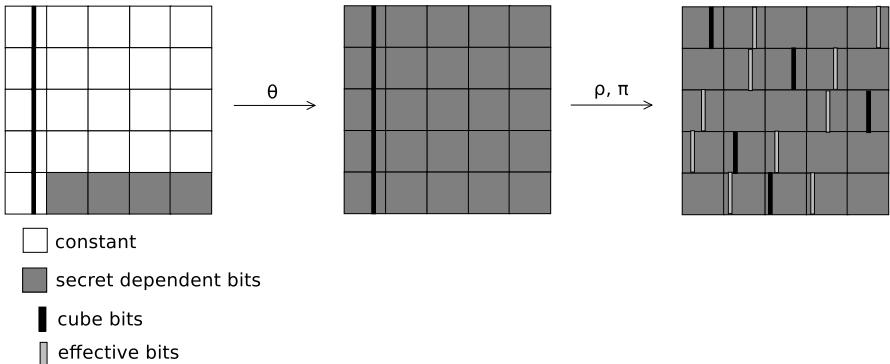


Fig. 8. Example placement of the cube and effective bits before the first χ is applied

during the preprocessing phase, we enumerate and store the 2^{80} cube sums for all the possible values of the secret expressions in time $2^{80+32} = 2^{112}$, using 2^{80} memory. During the online phase, we simply request the outputs for the chosen cube, calculate the cube sums and compare with the values stored in memory. This online procedure recovers the secret expressions in 2^{32} data and time.

In order to recover all the 252 secret variables, we use a total of 8 cubes, obtained by rotating the variables of the initial cube inside the lanes by multiples of 8 towards the MSB (e.g., the second cube contains bits 8–16 of the lanes with $x = 0$). Each such cube changes the effective secret expressions that are multiplied with the cube variables (although their number remains 80). One can verify that the secret expressions multiplied with these 8 cubes contain sufficient information to recover all the 252 secret variables (by solving a system of linear equations in the 252 variables). Note that as we only have 252 secret variables, after exploiting the first few cubes, the values of some secret linear expressions is already determined, and this can be used to slightly optimize the attack.

Balanced Attack. As in the case of the Keccak MAC, we can use auxiliary variables to balance the preprocessing and online complexities, reaching the lower total complexity. In the preprocessing, we calculate and store only 2^{40} cube sums — a substantially smaller subset of all possible 2^{80} cube sums for the 80 effective secret expressions. Thus, the preprocessing time complexity is reduced to $2^{40+32} = 2^{72}$ and the memory complexity is reduced to 2^{40} .

During the online phase, we exploit the large freedom in the message bits (we have $1348 - 40 - 2 = 1306$ free message bits that are not cube variables or padding bits) to set auxiliary variables that affect the values of the 80 effective bits which are multiplied with the cube variables. Then, we request the plaintexts and calculate online the cube sums for 2^{40} (unknown beforehand) different values of these 80 effective bits. According to the birthday paradox, with high probability, the (unknown beforehand) values of the 80 effective bits, in one of these online trials, will match one of their 2^{40} preprocessed values. This match will be detected by equating the cube sums, and allow us to recover the 80 effec-

tive secret expressions. Therefore, the data and time complexities of recovering 80 effective secret expressions are $2^{32+40} = 2^{72}$, and including preprocessing, the total time complexity is $2 \cdot 2^{72} = 2^{73}$.

In order to recover all the 252 secret variables, we use the 8 cubes defined in the basic (non-balanced) attack. Therefore, the total time complexity of the attack is $8 \cdot 2^{73} = 2^{76}$, the data complexity is $8 \cdot 2^{72} = 2^{75}$ and it requires $8 \cdot 2^{40} = 2^{43}$ words of memory (the memory can be reduced to 2^{40} if the cubes are analysed sequentially).

There are many possible tradeoffs between complexities of the preprocessing and the online phase. Interesting parameters are obtained by using only 24 auxiliary variables. In this case, according to the birthday paradox, we need to iterate over $2^{80-24} = 2^{56}$ values of the effective secret expressions for each cube during the preprocessing. Thus, the preprocessing complexity is $8 \cdot 2^{32+56} = 2^{91}$, the memory complexity is $8 \cdot 2^{56} = 2^{59}$, while the data and online time complexities are $8 \cdot 2^{32+24} = 2^{59}$ as well.

7 Conclusion

We mounted various types of algebraic attacks on keyed Keccak variants, breaking up to 6 rounds with practical complexity, and up to 9 rounds much faster than the exhaustive search. Our attacks incorporate in a novel way both algebraic and structural analysis of Keccak. We expect that the techniques developed in this paper will be further refined and used in future analysis of Keccak and related designs.

Considering attacks that break core security properties of the keyless, hashing mode, much faster than exhaustive search, the best result is 5 rounds [13]. As we can break up to 9 rounds of the keyed variants, the conclusion from our analysis is that the security margin of Keccak is somewhat reduced in the keyed modes. However, the full 24-round variants still have a big security margin. For Keyak – the authenticated encryption scheme based on Keccak – the nominal number of rounds is 12 and we showed that its security margin is smaller (but still sufficient).

Acknowledgments. Project was financed by Polish National Science Centre, project DEC-2013/09/D/ST6/03918. Josef Pieprzyk was supported by the ARC grant DP0987734.

Appendix

A

Table 2. An example of a cube and corresponding superpolys used in the attack on 5-round Keccak MAC

cube: 128,130,131,139,145,146,147,148,151,155,158,160,161,163,164,165,185,186,189, 190,193,196,205,212, 220,225,229,238,242,245,249			
superpoly	output bit	superpoly	output bit
x_{77}	7	$1 + x_{110}$	13
$1 + x_{113}$	15	x_{25}	31
$1 + x_{103}$	42	$1 + x_{105}$	69
x_{44}	84	x_{123}	87
$1 + x_{100}$	96	$1 + x_{104}$	100
x_{17}	112	$x_{38} + x_{51}$	71
$1 + x_7 + x_{19}$	91	$1 + x_{80} + x_{122}$	113
$x_{17} + x_{68} + x_{116}$	114		

B

Table 3. An example of a cube and corresponding superpolys found for 5.5 rounds, used in the attack on the 6-round Keccak working in the stream cipher mode

cube: 128,133,134,137,138,145,153,154,155,157,158,161,175,180,182,187,191,192,195, 199,206,208,211,220,227,229,245,247,249,251,252			
superpoly	output bit	superpoly	output bit
x_{76}	1	$1 + x_{64}$	13
x_{41}	17	x_{106}	28
$1 + x_{85}$	38	$1 + x_{32}$	46
$1 + x_{10}$	49	x_0	70
x_{109}	71	$1 + x_{121}$	73
$1 + x_{25}$	88	x_{96}	91
$1 + x_{35}$	95	$1 + x_{68}$	97
x_{42}	106	x_{72}	111
x_{26}	112	$1 + x_{34}$	123
x_{116}	125		

C

Table 4. An example of a cube and corresponding superpolys found for the 5.5-round variant with the reduced (400-bit) state. Cubes were used in the attack on the 6-round Keccak MAC.

cube: 80,82,84,85,87,90,91,96,102,105,109,110,111,116,119,122,128,130,133,134,136, 139,140,141,145,146, 147,149,153,156,159			
superpoly	output bit	superpoly	output bit
$1 + x_1 + x_2 + x_8 + x_{11} + x_{12}$ $+ x_{16} + x_{17} + x_{18} + x_{19} + x_{20}$ $+ x_{31} + x_{35} + x_{37} + x_{40} + x_{41}$ $+ x_{50} + x_{52} + x_{62} + x_{65} + x_{69}$ $+ x_{71} + x_{74} + x_{79}$	29	$x_2 + x_4 + x_5 + x_{16} + x_{17} +$ $x_{20} + x_{22} + x_{24} + x_{28} + x_{34} +$ $x_{40} + x_{42} + x_{43} + x_{44} + x_{47} +$ $x_{49} + x_{51} + x_{52} + x_{53} + x_{54} +$ $x_{56} + x_{60} + x_{61} + x_{62} + x_{67} +$ $x_{69} + x_{72} + x_{73} + x_{75} + x_{78}$	98
$x_0 + x_2 + x_4 + x_7 + x_8 + x_{10}$ $+ x_{11} + x_{13} + x_{14} + x_{16} + x_{17}$ $+ x_{20} + x_{23} + x_{26} + x_{28} + x_{30}$ $+ x_{31} + x_{32} + x_{34} + x_{35} + x_{36}$ $+ x_{39} + x_{41} + x_{43} + x_{46} + x_{49}$ $+ x_{52} + x_{54} + x_{56} + x_{63} + x_{76}$	79		

D

In this section, we provide detailed analysis of some elements of the divide-and-conquer attack of Section 6.

Proofs of Properties 1 and 2

We prove the two properties on which the basic attack of Section 6 is based. Recall that we select the cube variables v_1, v_2, \dots, v_{32} in $A[2, 2]$ and $A[2, 3]$, such that the column parities remain constant for all the 2^{32} possible values of the variables.

Property 1 (restated). *The cube sum of each output bit after 6 rounds does not depend on the value of $A[1, 0]$.*

Proof. We fix the value of $A[0, 0]$ to an arbitrary constant, and symbolically represent the 64 bits of $A[1, 0]$ as secret variables. We track the symbolic evolution of the 64 secret variables and 32 public variables throughout the first round: Due to θ , the secret variables of $A[1, 0]$ linearly diffuse to $A[0, *]$, $A[2, *]$ and $A[1, 0]$, while the cube variables of $A[2, 2]$ and $A[2, 3]$ do not diffuse (as the column parities of $A[2, *]$ remain constant). Then, ρ rotates the lanes, but does not effect the inter-lane diffusion. The mapping π reorders the lanes, and after its application, the bits of the 2 lanes $A[2, 0], A[3, 3]$ linearly depend on the public variables, while the bits of the following 11 lanes linearly depend on the secret variables. Figure 9 shows the details.

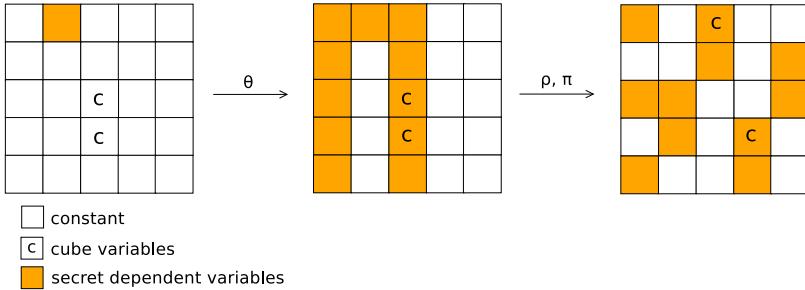


Fig. 9. Diffusion of the secret dependent variables

After the application of the linear mappings, we apply χ to the state. Despite its non-linearity, the only non-linear operation of χ is multiplying together bits in consecutive lanes (with the same y and z indexes). Thus, the bits of the 2 lanes that depend on the public variables, $A[2, 0], A[3, 3]$, are only multiplied with the bits of $A[1, 0], A[3, 0], A[2, 3], A[4, 3]$. Since these 4 lanes are constants (they do not depend on any cube or secret variables), then the cube variables are not multiplied by any variables throughout the first round (ι is a linear mapping which does not change this property). In other words, the symbolic form of each state bit $A[x, y, z]$ can be written as $L_{x,y,z}(v_1, v_2, \dots, v_{32}, w_1, w_2, \dots)$, where $L_{x,y,z}$ is some linear function, and the variables w_i depend only on the secret variables (and not on the cube variables v_1, v_2, \dots, v_{32}).

We now analyse the symbolic form of the state bits after 6 Keccak rounds, whose algebraic degree in the state variables after one round is $2^5 = 32$. Given the special symbolic form of the state bits after one Keccak round, the degree of each state bit after 6 rounds in the variables $v_1, v_2, \dots, v_{32}, w_1, w_2, \dots$ is at most 32, and thus the superpoly of the monomial $v_1 v_2 \dots v_{32}$ is constant. As a result, the cube sum of each state bit after 6 rounds is a constant, which does not depend on the value of the secret variables. This proves Property 1.

Property 2 (restated). *The cube sums of the output bits after 6 rounds depend on the value of $A[0, 0]$.*

Proof. When considering the bits of $A[0, 0]$ as secret variables, they are multiplied with the cube variables in the first round (e.g. $A[0, 0]$ diffuses to $A[1, 0]$, whose bits are multiplied with $A[2, 0]$ due to χ). After 6 rounds, we expect the degree of the output bits, in the state bits after one round, to be $2^5 = 32$. Consequently, we expect the superpoly of $v_1 v_2 \dots v_{32}$ for an output bit to generally depend on the value of $A[0, 0]$, and thus the cube sums of the output bits generally depend on the value of $A[0, 0]$. This proves Property 2.

Simulation Results

In our 6- and 7-round key recovery attack, the most desired situation is when each 64-bit key would correspond to a distinct vector of cube sums. However, checking all 2^{64} cases, where each case requires summing over 2^{32} messages,

is infeasible. Therefore, we conducted experiments checking a limited number of keys and using smaller cubes. First, we checked 2^{16} randomly chosen keys using 16-bit cube and nearly all keys have their unique cube sum vector. Only a very small fraction (below 0.007%) share the output vector with other keys. The second experiment, ran on the smaller variant with 400-bit state, with 2^{16} randomly chosen keys, showed that each key has its unique cube sum vector. Thus, our simulation results is a strong indication that assumptions taken for the 6- and 7-round key recovery attacks are sound.

References

1. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. <http://competitions.cr.yp.to/caesar.html>
2. Aumasson, J.P., Meier, W.: Zero-sum Distinguishers for Reduced Keccak-f and for the Core Functions of Luffa and Hamsi. Tech. rep., NIST mailing list (2009)
3. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and trivium. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009)
4. Bard, G.V., Courtois, N.T., Nakahara Jr., J., Sepehrdad, P., Zhang, B.: Algebraic, AIDA/cube and side channel analysis of KATAN family of block ciphers. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 176–196. Springer, Heidelberg (2010)
5. Bellare, M., Canetti, R., Krawczyk, H.: Message Authentication Using Hash Functions: the HMAC Construction. *CryptoBytes* **2**(1), 12–15 (1996)
6. Bernstein, D.J.: Second Preimages for 6 (7? (8??)) Rounds of Keccak? NIST mailing list (2010). <http://ehash.iakr.tugraz.at/uploads/6/65/NIST-mailing-list-Bernstein-Daemen.txt>
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic Sponges. <http://sponge.noekeon.org/CSF-0.1.pdf>
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak Sponge Function Family Main Document. <http://keccak.noekeon.org/Keccak-main-2.1.pdf>
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: Keyak. <http://keyak.noekeon.org>
10. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the Sponge: Single-pass Authenticated Encryption and Other Applications. *Cryptology ePrint Archive*, Report 2011/499 (2011). <http://eprint.iacr.org/>
11. Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of KECCAK and *Luffa*. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011)
12. Dinur, I., Dunkelman, O., Shamir, A.: New attacks on keccak-224 and keccak-256. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 442–461. Springer, Heidelberg (2012)
13. Dinur, I., Dunkelman, O., Shamir, A.: Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 219–240. Springer, Heidelberg (2014)
14. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)

15. Dinur, I., Shamir, A.: Breaking grain-128 with dynamic cube attacks. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 167–187. Springer, Heidelberg (2011)
16. Fischer, S., Khazaei, S., Meier, W.: Chosen IV statistical analysis for key recovery attacks on stream ciphers. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 236–245. Springer, Heidelberg (2008)
17. Homsirikamol, E., Morawiecki, P., Rogawski, M., Srebrny, M.: Security margin evaluation of SHA-3 contest finalists through SAT-based attacks. In: Cortesi, A., Chaki, N., Saeed, K., Wierzchoń, S. (eds.) CISIM 2012. LNCS, vol. 7564, pp. 56–67. Springer, Heidelberg (2012)
18. Jovanovic, P., Luykx, A., Mennink, B.: Beyond $2^{c/2}$ Security in Sponge-Based Authenticated Encryption Modes. Cryptology ePrint Archive, Report 2014/373 (2014)
19. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of NLFSR-based cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer, Heidelberg (2010)
20. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communications and Cryptography. The Springer International Series in Engineering and Computer Science, vol. 276, pp. 227–233. Springer, US (1994)
21. Lathrop, J.: Cube Attacks on Cryptographic Hash Functions. Master's thesis, Rochester Institute of Technology (2009)
22. Naya-Plasencia, M., Röck, A., Meier, W.: Practical analysis of reduced-round KEC-CAK. In: Bernstein, D.J., Chatterjee, S. (eds.) INDOCRYPT 2011. LNCS, vol. 7107, pp. 236–254. Springer, Heidelberg (2011)
23. Vielhaber, M.: Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. Cryptology ePrint Archive, Report 2007/413 (2007)

Twisted Polynomials and Forgery Attacks on GCM

Mohamed Ahmed Abdelraheem^(✉), Peter Beelen, Andrey Bogdanov,
and Elmar Tischhauser

Department of Mathematics and Computer Science,
Technical University of Denmark, Kongens Lyngby, Denmark
`{mohab, pabe, anbog, ewti}@dtu.dk`

Abstract. Polynomial hashing as an instantiation of universal hashing is a widely employed method for the construction of MACs and authenticated encryption (AE) schemes, the ubiquitous GCM being a prominent example. It is also used in recent AE proposals within the CAESAR competition which aim at providing nonce misuse resistance, such as POET. The algebraic structure of polynomial hashing has given rise to security concerns: At CRYPTO 2008, Handschuh and Preneel describe key recovery attacks, and at FSE 2013, Procter and Cid provide a comprehensive framework for forgery attacks. Both approaches rely heavily on the ability to construct *forgery polynomials* having disjoint sets of roots, with many roots (“weak keys”) each. Constructing such polynomials beyond naïve approaches is crucial for these attacks, but still an open problem.

In this paper, we comprehensively address this issue. We propose to use *twisted polynomials* from Ore rings as forgery polynomials. We show how to construct sparse forgery polynomials with full control over the sets of roots. We also achieve complete and explicit disjoint coverage of the key space by these polynomials. We furthermore leverage this new construction in an improved key recovery algorithm.

As cryptanalytic applications of our twisted polynomials, we develop the first universal forgery attacks on GCM in the weak-key model that do not require nonce reuse. Moreover, we present universal weak-key forgeries for the nonce-misuse resistant AE scheme POET, which is a CAESAR candidate.

Keywords: Authenticated encryption · Polynomial hashing · Twisted polynomial ring (Ore ring) · Weak keys · GCM · POET

1 Introduction

Authenticated encryption (AE) schemes are symmetric cryptographic primitives combining the security goals of confidentiality and integrity. Providing both ciphertext and an authentication tag on input of a plaintext message, they allow

Due to page limitations, several details are omitted in this proceedings version. A full version is available at [2].

two parties sharing a secret key to exchange messages in privacy and with the assurance that they have not been tampered with.

Approaches to construct AE schemes range from generic composition of a symmetric block or stream cipher for confidentiality and a message authentication code (MAC) for integrity to dedicated designs. An important method for constructing both stand-alone MACs and the authentication tag generation part of dedicated AE algorithms is based on universal hash functions, typically following the Carter-Wegman paradigm [21]. This construction enjoys information-theoretic security and is usually instantiated by *polynomial hashing*, that is, the evaluation of a polynomial in H (the authentication key) over a finite field with the message blocks as coefficients.

One of the most widely adopted AE schemes is the Galois Counter Mode (GCM) [6], which has been integrated into important protocols such as TLS, SSH and IPsec; and furthermore has been standardized by among others NIST and ISO/IEC. It combines a 128-bit block cipher in CTR mode of operation for encryption with a polynomial hash in \mathbb{F}_2^{128} over the ciphertexts to generate an authentication tag. The security of GCM relies crucially on the uniqueness of its nonce parameter [7, 10, 11].

As a field, authenticated encryption has recently become a major focus of the cryptographic community due to the ongoing CAESAR competition for a portfolio of recommended AE algorithms [1]. A large number of diverse designs has been submitted to this competition, and a number of the submissions feature polynomial hashing as part of their authentication functionality. Among these, the new AE schemes POET [3], Julius [5] and COBRA [4] feature stronger security claims about preserving confidentiality and/or integrity under nonce reuse (so-called *nonce misuse resistance* [12]).

Background. The usual method to build a MAC or the authentication component of an AE scheme from universal hash functions is to use *polynomial hashing*, in other words, to evaluate a polynomial in the authentication key with the message or ciphertext blocks as coefficients:

Definition 1 (Polynomial-based Authentication Scheme). *A polynomial hash-based authentication scheme processes an input consisting of a key H and plaintext/ciphertext $M = (M_1 || M_2 || \dots || M_l)$, where each $M_i \in \mathbb{F}_2^n$, by evaluating the polynomial*

$$h_H(M) := \sum_{i=1}^l M_i H^i \in \mathbb{F}_2^n.$$

To produce an authentication tag, the value $h_H(M)$ is often processed further, for example by encryption, or additive combination with another pseudorandom function. For a survey of existing constructions, we refer the reader to [10]. Out of these schemes, GCM [6, 13] is by far the most important and widespread algorithm. We therefore recapitulate existing security results about polynomial hashing at the example of GCM.

The Galois Counter Mode. GCM is defined as follows. It takes as input the plaintext $M = M_1 || M_2 || \dots || M_l$, a key k and a nonce N . It outputs corresponding ciphertext $C = C_1 || C_2 || \dots || C_l$ and an authentication tag T . The ciphertext blocks are generated using a block cipher E_k (usually AES) in counter mode: $C_i = E_k(J_{i-1}) \oplus M_i$, with J_0 an initial counter value derived from N , and the J_1, J_2, \dots successive increments of J_0 . The ciphertexts are then processed with polynomial hashing to generate the tag

$$T = E_k(J_0) \oplus h_H(C)$$

with $H = E_k(0)$ as the authentication (hash) key. GCM is typically instantiated with a 128-bit block cipher, uses 128-bit keys and 96-bit nonces and produces 128-bit tags.

Joux' “forbidden” attack. Soon after the proposal of GCM, Joux [11] pointed out that the security of GCM breaks down completely if nonces are re-used with the same key. Since GCM is built upon the assumption of nonce uniqueness, his attack is referred to as the “forbidden” attack against GCM. It recovers the hashing key H using pairs of different messages M and M' that are authenticated using the same nonce N . This leads to the following equation in one unknown H :

$$T \oplus T' = h_H(C) \oplus E_K(N) \oplus h_H(C') \oplus E_K(N) = h_H(C \oplus C'),$$

where C/C' and T/T' are the ciphertext/tag of M/M' . This is equivalent to saying that the polynomial $T \oplus T' \oplus h_H(C \oplus C')$ has a root at H . By using multiple message pairs and computing the GCD of the arising polynomials, H can be uniquely identified. This attack does not apply to the nonce-respecting adversarial model.

Ferguson's Short Tag attacks. While Joux' attack establishes GCM's sensitivity to nonce reuse, Ferguson [7] demonstrated that truncation of its output to shorter tags of $s < 128$ bits not only (generically) limits its authentication security level to $s/2$ bits, but also allows a key recovery attack with little more than $2^{s/2}$ queries, which especially does not require a collision on the full 128-bit polynomial hash. Ferguson's attacks make use of so-called *error polynomials*

$$\sum_{i=1}^l (C_i - C'_i) H^i,$$

with the C_i the original and the C'_i the modified ciphertext blocks. Since GCM operates in a field or characteristic two, squaring is a linear operation, and this allows Ferguson to consider *linearized error polynomials*, i.e. where only the coefficients of H^{2^i} are nonzero. The effect of these modifications on the first bits of the (truncated) authentication tag is then a linear function of H and the coefficients. Using linear algebra, the coefficients of the error polynomial are then computed such that the first $s/2$ bits of the shortened tag will not change. The attack then exploits a generic birthday-type collision on the remaining $s/2$ tag

bits to obtain a complete collision on the short tag. A small number of further forgeries then yield enough linear relations about bits of H to allow its complete recovery. Note that this attack does not require nonce reuse.

Handschuh and Preneel’s Key Recovery Attacks. Handschuh and Preneel [10] propose various methods for recovering the hash key of polynomial hashing-based MACs, among them GCM. The main idea is to obtain a valid ciphertext-tag pair C, T and then to attempt verification with a different message C' but the same tag; here C' is chosen such that $C - C'$ has many distinct roots. If verification is not successful, another C'' is used which is chosen such that $C - C''$ has no roots in common with $C - C'$, and so on. Once a verification succeeds, this indicates that the authentication key is among the roots of this polynomial. Further queries can then be made to subsequently reduce the search space until the key is identified. When using polynomials of degree d in each step, the total number of verification queries needed is $2^n/d$. Knowing the authentication key then allows the adversary to produce forgeries for any given combination of nonce and corresponding ciphertext blocks. The attack of [10] does not require nonce reuse, however is limited to ciphertexts as it does not allow the adversary to create universal forgeries for any desired *plaintext message*.

Handschuh and Preneel further identify the key $H = 0$ as a trivially weak key for GCM-like authentication schemes. They further provide a formalization of the concept of weak keys, namely a class D of keys is called *weak* if membership in this class requires less than $|D|$ key tests and verification queries.

Saarinen’s Cycling Weak Key Forgery Attacks. This concept of weak keys for polynomial authentication was taken a step further by Saarinen in [20], where a forgery attack for GCM is described for the case where the order of the hash key H in $\mathbb{F}_{2^{128}}^\times$ is small. If the hash key belongs to a cyclic subgroup of order t , i.e. $H^{t+1} = H$, then the attacker can create a blind forgery by simply swapping any two ciphertext blocks C_i and C_{i+jt} . Such hash keys with short cycles (small value of t) can be labelled as *weak keys*. In other words, Saarinen identifies all elements with less than maximal order in $\mathbb{F}_{2^{128}}^\times$ as weak keys. Since constructing a corresponding forgery requires a message length of at least 2^t blocks, and GCM limits the message to 2^{32} blocks, this means that all keys with order less than 2^{32} are weak keys for GCM. We finally note that cycling attacks depend on the factorisation of $2^n - 1$, since any subgroup order is a divisor of the order of $\mathbb{F}_{2^{128}}^\times$.

Procter and Cid’s General Weak-Key Forgery Framework. The idea behind cycling attacks was extended and formalized by Procter and Cid [15] by introducing the notion of so-called *forgery polynomials*: Let H be the (unknown) hash key. A polynomial $q(X) = \sum_{i=1}^l q_i X^i$ is then called a forgery polynomial if it has H as a root, i.e. $q(H) = 0$. This designation is explained by noting that for $C = (C_1 || C_2 || \dots || C_l)$ and writing $Q = q_1 || \dots || q_l$, we have

$$h_H(C) = h_H(C + Q),$$

that is, adding the coefficients of q yields the same authentication tag, i.e. a forgery.¹ More concretely, for GCM, we have that $(N, C + Q, T)$ is a forgery for (N, C, T) whenever $q(H) = 0$. This also means that all roots of q can be considered weak keys in the sense of [10]. In order to obtain forgeries with high probability, Procter and Cid note that a concrete choice for q should have a high degree and preferably no repeated roots.

Since any choice of q is a forgery polynomial for its roots as the key, Procter and Cid establish the interesting fact that *any* set of keys in polynomial hashing can be considered weak: membership to a weak key class D can namely be tested by one or two verification queries using the forgery polynomial $q(X) = \prod_{d \in D} (X - d)$ regardless of the size of D . They also note that such a forgery polynomial can be combined with the key recovery technique of [10], namely by using the polynomial $q(X) = \prod_{H \in \mathbb{F}_2^n, H_n=0} (X - H)$ and then subsequently fixing more bits of H according to the results of the verification queries. This only requires two queries for a first forgery, and at most $n + 1$ for complete key recovery. Note however that this requires messages lengths up to 2^n blocks, which is clearly infeasible for GCM (where $n = 128$).

We also note that all previously described attacks can be seen as special cases of Procter and Cid's general forgery framework [15, 16].

Our Problem. We start by noting that besides the attacks of Joux and Ferguson, which apply to the special cases where the nonce is reused or tags are truncated, only Saarinen's cycling attack gives a concrete security result on GCM and similar authentication schemes. In the formalism of [15], it uses the forgery polynomials $X^t - X$ with $t < 2^{32}$ the subgroup order. To the best of our knowledge, no other explicit forgery polynomials have been devised. In [15], two generic classes of forgery polynomials are discussed: random polynomials of degree d in $\mathbb{F}_{2^n}[X]$ or naïve multiplication of linear factors $(x - H_1) \cdots (x - H_d)$. The latter construction requires d multiplications already for the *construction* of the forgery polynomial, which quickly becomes impractical. We also note that in both cases, the coefficients will be “dense”, i.e. almost all of them will be nonzero. This means that all of the ciphertext blocks have to be modified by the adversary to submit each verification query. In the same sense, the observation of [15] that any key is weak is essentially a certificational result only since $|D|$ multiplications are needed to produce q for a weak key class of size $|D|$. The construction of explicit forgery polynomials is left as an important open problem in [15].

Similarly, the key recovery technique of [10] does not deal with the important question of how to construct new polynomials of degree d having distinct roots from all previously chosen ones, especially without the need to store all d roots from each of the $2^n/d$ iterations. These observations lead to the following questions:

¹ Note that forgery polynomials are conceptually different from Ferguson's error polynomials, since the authentication key H typically is not a root of an error polynomial, while this is the defining property for forgery polynomials.

Can we efficiently construct explicit forgery polynomials having prescribed sets of roots, ideally having few nonzero coefficients? Moreover, can we disjointly cover the entire key space using these explicit forgery polynomials?

Answers to these questions would essentially solve the open problem mentioned in [15], and also make the observation concrete that *any* key in polynomial hashing can be considered weak. It would also improve the key recovery algorithm of Handschuh and Preneel [10]. On the application side, we ask whether *plaintext-universal forgeries* for GCM can be constructed in the *nonce-respecting adversarial model*.

Our Results. In this paper, we answer the above-mentioned questions in the affirmative. We comprehensively address the issue of polynomial construction and selection in forgery and key recovery attacks on authentication and AE schemes based on polynomial hashing. In detail, the contributions of this paper are as follows.

Explicit construction of sparse forgery polynomials. In contrast to the existing generic methods to construct forgery polynomials, we propose a construction based on so-called twisted polynomial rings that allows us to explicitly describe polynomials of degree 2^d in any finite field \mathbb{F}_2^n which have as roots precisely the elements of an arbitrary d -dimensional subspace of \mathbb{F}_2^n , independent of n or the factorisation of $2^n - 1$. While achieving this, our polynomials are very sparse, having at most $d + 1$ nonzero coefficients.

Complete disjoint coverage of the key space by forgery polynomials. In order to recover the authentication key (as opposed to blind forgeries), the attacks of Handschuh and Preneel [10] and Procter and Cid [15] need to construct polynomials having a certain set of roots, being disjoint from the roots of all previous polynomials. We propose an explicit algebraic construction achieving the partitioning of the whole key space \mathbb{F}_2^n into roots of structured and sparse polynomials. This substantiates the certificational observation of [15] that any key is weak, in a concrete way. We give an informal overview of our construction of twisted forgery polynomials in the following proposition.

Proposition (informal). *Let $q = r^e$ and let V be a subspace of \mathbb{F}_q of over the field \mathbb{F}_r of dimension d . Then there exists a twisted polynomial ϕ from the Ore ring $\mathbb{F}_q\{\tau\}$ with the following properties:*

1. *ϕ can be written as $\phi(X) = c_0 + \sum_{i=1}^d c_i X^{2^i}$, i.e. ϕ has at most $d+1$ nonzero coefficients;*
2. *For any $a \in \mathbb{F}_q$, the polynomial $\phi(X) - \phi(a)$ has exactly $a + V$ as set of roots;*
3. *The sets of roots of the polynomials $\phi(X) - b$ with $b \in \text{Im } \phi$ partition \mathbb{F}_q .*

Improved key recovery algorithm. We then leverage the construction of sparse forgery polynomials from the twisted polynomial ring to propose an improved key recovery algorithm, which exploits the particular structure of the root spaces of our forgery polynomials. In contrast to the key recovery techniques of [10] or [15], it only requires the modification of a logarithmic number of message blocks in each iteration (i.e., d blocks for a 2^d -block message). It also allows arbitrary trade-offs between message lengths and number of queries.

New universal forgery attacks on GCM. Turning to applications, we develop the first universal forgery attacks on GCM in the weak-key model that do not require nonce reuse. We first use tailored twisted forgery polynomials to recover the authentication key. Depending on the length of the nonce, we then either use a sliding technique on the counter encryptions or exploit an interaction between the processing of different nonce lengths to obtain valid ciphertext-tag pairs for any given combination of nonce and plaintext.

Analysis of POET, Julius, and COBRA. Using our framework, we finally present further universal forgery attacks in the weak-key model also for the recently proposed nonce-misuse resistant AE schemes POET, Julius, and COBRA.

Our results on POET prompted the designers to formally withdraw the variant with finite field multiplications as universal hashing from the CAESAR competition. Previously, an error in an earlier specification of POET had been exploited for constant-time blind forgeries [9]. This attack however does not apply to the corrected specification of POET. Likewise, for COBRA, a previous efficient attack by Nandi [14] does not yield universal forgeries.

Organization. The remainder of the paper is organized as follows. We introduce some common notation in Sect. 2. In Sect. 3, we describe our method to construct explicit and sparse forgery polynomials. Sect. 4 proposes two approaches to construct a set of explicit forgery polynomials whose roots partition the whole finite field \mathbb{F}_2^{128} . In Sect. 5, we describe our improved key recovery algorithm. In Sect. 6, two universal weak-key forgery attacks against GCM are presented. In Sect. 7, we present several universal forgery attacks on POET under the weak-key assumption. For the attacks on Julius and COBRA, we refer to the full version of this paper [2]. We conclude in Sect. 8.

2 Preliminaries

Throughout the paper, we denote by \mathbb{F}_{p^n} the finite field of order p^n and characteristic p , and write \mathbb{F}_p^n for the corresponding n -dimensional vector space over \mathbb{F}_p . We use $+$ and \oplus interchangeably to denote addition in \mathbb{F}_{2^n} and \mathbb{F}_2^n .

Forgery polynomials. We formally define forgery polynomials [15] as polynomials $q(X) = \sum_{i=1}^r q_i X^i$ with the property that that $q(H) = 0$ for the authentication key H . Assume that $M = (M_1 || M_2 || \cdots || M_l)$ and that $l \leq r$. Then

$$h_H(M) = \sum_{i=1}^r M_i H^i = \sum_{i=1}^l M_i H^i + \sum_{i=1}^r q_i H^i = \sum_{i=1}^r (M_i + q_i) H^i = h_H(M + Q)$$

where $Q = q_1 || \cdots || q_r$. If $l < r$, we simply pad M with zeros. Throughout the paper, we will refer to Q as the binary coefficient string of a forgery polynomial $q(X)$.

Using q as a forgery polynomial in a blind forgery gives a success probability $p = \frac{\#\text{roots of } q(X)}{2^n}$. Therefore, in order to have a forgery using the polynomial $q(X)$ with high probability, $q(X)$ should have a high degree and preferably no repeated roots.

In the next section, we will present methods to construct explicit sparse forgery polynomials $q(X)$ with distinct roots and high forgery probability.

3 Explicit Construction of Twisted Forgery Polynomials

When applying either the key recovery attack of [10] or any of the forgery or key recovery attacks of [15], a crucial issue lies in the selection of polynomials that have a certain number ℓ of roots in \mathbb{F}_2^n , and additionally being able to select each polynomial to have no common roots with the previous ones. Ideally, these polynomials should both be described by explicit constructive formulas, and they should be sparse, i.e. have few nonzero coefficients.

As noted in [15], the direct way to do this is to choose distinct elements $\alpha_1, \dots, \alpha_\ell \in \mathbb{F}_{2^n}$ and to work out the product $(X - \alpha_1) \cdots (X - \alpha_\ell)$, which quickly gets impractical for typical values of ℓ and will not result in sparse polynomials. The second suggestion described in [15] is to select them at random, which is efficient, but also does not produce sparse polynomials. Moreover, as noted in [16], subsequently chosen random polynomials will likely have common roots, which rules out the key recovery attacks of both [10] and [16].

The only proposed explicit construction of forgery polynomials so far are the polynomials $X^t - 1$ with $t|(2^{128} - 1)$, due to Saarinen [20]. Their roots correspond precisely to the cyclic subgroups of \mathbb{F}_2^{128} , which also limits their usefulness in the key recovery attacks.

In this section, we propose a new method which yields explicit constructions for polynomials with the desired number of roots. At the same time, the resulting polynomials are sparse in the sense that a polynomial with 2^d roots will have at most $d + 1$ nonzero coefficients.

For this, we use the fact that $\mathbb{F}_{2^{128}}$ can be seen as a vector space (of dimension 128) over \mathbb{F}_2 . More precisely, given a subvector space V of $\mathbb{F}_{2^{128}}$ of dimension d with basis $\{b_1, \dots, b_d\}$, we describe a fast procedure to find a polynomial $p_V(X) \in \mathbb{F}_{2^{128}}[X]$ whose roots are exactly all elements of V . Note that this implies that $\deg p_V(X) = 2^d$. We will also see that the $p_V(X)$ is sparse, more

precisely that the only coefficients of $p_V(X)$ that may be distinct from zero are the coefficients of the monomials X^{2^i} with $0 \leq i \leq d$. In particular this will imply that $p_V(X)$ has at most $d + 1$ non-zero coefficients despite the fact that it has degree 2^d .

To explain the above, we introduce the concept of a twisted polynomial ring, also called an Ore ring.

Definition 2. Let \mathbb{F}_q be a field of characteristic p . The *twisted polynomial or Ore ring* $\mathbb{F}_q\{\tau\}$ is defined as the set of polynomials in the indeterminate τ having coefficients in \mathbb{F}_q with the usual addition, but with multiplication defined by the relation $\tau\alpha = \alpha^p\tau$ for all $\alpha \in \mathbb{F}_q$.

The precise ring we will need is the ring $\mathbb{F}_{2^{128}}\{\tau\}$. In other words, two polynomials in τ can be multiplied as usual, but when multiplying the indeterminate with a constant, the given relation applies. This makes the ring a non-commutative ring (see [8] for an overview of some of its properties). One of the reasons to study this ring is that it gives a convenient way to study linear maps from $\mathbb{F}_{2^{128}}$ to itself, when viewed as a vector space over \mathbb{F}_2 . A constant $\alpha \in \mathbb{F}_{2^{128}}\{\tau\}$ then corresponds to the linear map sending $x \in \mathbb{F}_{2^{128}}$ to $\alpha \cdot x$, while the indeterminate τ corresponds to the linear map sending $x \in \mathbb{F}_{2^{128}}$ to x^2 . Addition in the Ore ring corresponds to the usual addition of linear maps, while multiplication corresponds to composition of linear maps. This explains the relation $\tau \cdot \alpha = \alpha^2 \cdot \tau$, since both expressions on the left and right of the equality sign correspond to the linear map sending x to $\alpha^2 x^2$. To any element ϕ from the Ore ring, we can associate a polynomial $\phi(X)$, by replacing τ^i with X^{2^i} . The resulting polynomials have possibly non-zero coefficients from $\mathbb{F}_{2^{128}}$ only for those monomials X^e , such that e is a power of 2. Such polynomials are called linearized and are just yet another way to describe linear maps from $\mathbb{F}_{2^{128}}$ to itself. The advantage of this description is that the null space of a linear map represented by a linearized polynomial $p(X)$ just consists of the roots of $p(X)$ in $\mathbb{F}_{2^{128}}$.

Now we describe how to find a polynomial $p_V(X)$ having precisely the elements of a subspace V of $\mathbb{F}_{2^{128}}$ as roots. The idea is to construct a linear map from $\mathbb{F}_{2^{128}}$ to itself having V as null space recursively. We will assume that we are given a basis $\{\beta_1, \dots, \beta_d\}$ of V . For convenience we define V_i to be the subspace generated by $\{\beta_1, \dots, \beta_i\}$. Note that $V_0 = \emptyset$ and $V_d = V$. Then we proceed recursively for $0 \leq i \leq d$ by constructing a linear map ϕ_i (expressed as an element of the Ore ring) with null space equal to V_i . For $i = 0$ we define $\phi_0 := 1$, while for $i > 0$ we define $\phi_i := (\tau + \phi_{i-1}(\beta_i))\phi_{i-1}$. For $d = 2$, we obtain for example

$$\phi_0 = 1, \quad \phi_1 = \tau + \beta_1$$

and

$$\phi_2 = (\tau + (\beta_2^2 + \beta_1\beta_2))(\tau + \beta_1) = \tau^2 + (\beta_2^2 + \beta_1\beta_2 + \beta_1^2)\tau + \beta_1\beta_2^2 + \beta_1^2\beta_2.$$

The null spaces of these linear maps are the roots of the polynomials

$$X, \quad X^2 + \beta_1 X$$

and

$$X^4 + (\beta_2^2 + \beta_1\beta_2 + \beta_1^2)X^2 + (\beta_1\beta_2^2 + \beta_1^2\beta_2)X.$$

It is easy to see directly that the null spaces of ϕ_0, ϕ_1, ϕ_2 have respective bases $\emptyset, \{\beta_1\}$ and $\{\beta_1, \beta_2\}$. More general, a basis for the null space of ϕ_i is given by $\{\beta_1, \dots, \beta_i\}$: indeed, since $\phi_i := (\tau + \phi_{i-1}(\beta_i))\phi_{i-1}$, it is clear that the null space of ϕ_{i-1} is contained in that of ϕ_i . Moreover, evaluating ϕ_i in β_i , we find that

$$\phi_i(\beta_i) = (\tau + \phi_{i-1}(\beta_i))(\phi_{i-1}(\beta_i)) = \phi_{i-1}(\beta_i)^2 + \phi_{i-1}(\beta_i)\phi_{i-1}(\beta_i) = 0.$$

This means that the null space of ϕ_i at least contains V_i (and therefore at least 2^i elements). On the other hand, the null space of ϕ_i can be expressed as the set of roots of the linearized polynomial $\phi_i(X)$, which is a polynomial of degree 2^i . Therefore the null space of ϕ_i equals V_i . For $i = d$, we obtain that the null space of ϕ_d is V . In other words: the desired polynomial $p_V(X)$ is just the linearized polynomial $\phi_d(X)$. The above claim about the sparseness of $p_V(X)$ now also follows. It is not hard to convert the above recursive description to compute $p_V(X)$ into an algorithm (see Alg. 5.1). In a step of the recursion, the multiplication $(\tau + \phi_{i-1}(\beta_i))\phi_{i-1}$ needs to be carried out in the Ore ring. Since the left term has degree one in τ , this is easy to do. To compute the coefficients in ϕ_i of all powers of τ one needs the commutation relation $\tau\alpha = \alpha^2\tau$ for $\alpha \in \mathbb{F}_{2^{128}}$. Computing a coefficient of a power of τ in a step of the recursion, therefore takes one multiplication, one squaring and one addition. The computation of ϕ_d can therefore be carried out without further optimization in quadratic complexity in d . A straightforward implementation can therefore be used to compute examples. Two examples are given in Appendix A with $d = 31$ and $d = 61$ needed for attacking GCM and POET.

Note that the above theory can easily be generalized to the setting of a finite field \mathbb{F}_{r^e} and \mathbb{F}_r -subspaces V over the field \mathbb{F}_{r^e} . In the corresponding Ore ring $\mathbb{F}_{r^e}\{\tau\}$ the commutation relation is $\tau\alpha = \alpha^r\tau$. Similarly as above, for any subspace of a given dimension d one can find a polynomial $p_V(X)$ of degree r^d having as set of roots precisely the elements of V . It may have non-zero coefficients only for monomials of the form X^{r^i} . In the program given in the full version of this paper [2], r and e can be chosen freely. See [8] for a more detailed overview of properties of linearized polynomials and the associated Ore ring.

4 Disjoint Coverage of the key Space with Roots of Structured Polynomials

The purpose of this section is to describe how one can cover the elements of a finite field \mathbb{F}_q by sets of roots of families of explicitly given polynomials. We will focus our attention to the case that $q = 2^{128}$, but the given constructions can directly be generalized to other values of $q = r^e$. We denote by γ a primitive element of \mathbb{F}_q . Two approaches will be described. The first one exploits the multiplicative structure of $\mathbb{F}_q \setminus \{0\}$, while the second one exploits the additive structure of \mathbb{F}_q seen as a vector space over \mathbb{F}_2 . We will in fact describe a way

to partition the elements of \mathbb{F}_q as sets of roots of explicit polynomials, that is to say that two sets of roots of distinct polynomials will have no elements in common. In both cases the algebraic fact that will be used is the following: Let G be a group with group operation $*$ and let $H \subset G$ be a subgroup. Then two cosets $g * H$ and $f * H$ are either identical or disjoint. Moreover the set of cosets gives rise to a partition of G into disjoint subsets.

4.1 Using the Multiplicative Structure

We first consider the group $G = \mathbb{F}_q \setminus \{0\}$ with group operation $*$ the multiplication in \mathbb{F}_q . For any factorization $q - 1 = n \cdot m$ we find a subgroup $H_m := \{\gamma^{nj} \mid 0 \leq j \leq m - 1\}$ consisting of m elements. This gives rise to the following proposition:

Proposition 1. *Let γ be a primitive element of the field \mathbb{F}_q and suppose that $q - 1 = n \cdot m$ for positive integers n and m . For i between 0 and $n - 1$ define*

$$A_i := \{\gamma^{i+nj} \mid 0 \leq j \leq m - 1\}.$$

Then the sets A_0, \dots, A_{n-1} partition $\mathbb{F}_q \setminus \{0\}$. Moreover, the set A_i consists exactly of the roots of the polynomial $X^m - \gamma^{im}$.

Proof. As mentioned we work in the multiplicative group $\mathbb{F}_q \setminus \{0\}$ and let H_m be the subgroup of G of order m . Note that $A_0 = H_m$ and that H_m is the kernel of the group homomorphism $\phi : G \rightarrow G$ sending x to x^m . In particular, H_m is precisely the set of roots of the polynomial $X^m - 1$. Any element from the coset gH_m is sent by ϕ to g^m . This means that gH_m is precisely the set of roots of the polynomial $X^m - g^m$. Note that $g^m = \gamma^{im}$ for some i between 0 and $n - 1$, so that the set of roots of $X^m - g^m$ equals $\gamma^i H_m = A_i$ for some i between 0 and $n - 1$. Varying i we obtain all cosets of H_m , so the result follows.

If $q = r^e$ for some prime power r , one can choose $n = r - 1$ and $m = r^{e-1} + \dots + r + 1$. For any element $\alpha \in \mathbb{F}_q$ we then have $\alpha^m \in \mathbb{F}_r$, since α^m is just the so-called $\mathbb{F}_q/\mathbb{F}_r$ -norm of α . Therefore the family of polynomials in the above lemma in this case take the particularly simple form $x^m - a$, with $a \in \mathbb{F}_r \setminus \{0\}$. In case $q = 2^{128}$, Proposition 1 gives rise to a family of polynomials whose roots partition $\mathbb{F}_{2^{128}} \setminus \{0\}$. For more details about the explicit form of these polynomials, we refer the reader to the full version [2] of this paper.

4.2 Using the Additive Structure

Now we use a completely different approach to partition the elements from \mathbb{F}_q in disjoint sets where we exploit the additive structure. Suppose again that $q = r^e$, then we can view \mathbb{F}_q as a vector space over \mathbb{F}_r . Now let $V \subset \mathbb{F}_q$ be any linear subspace (still over the field \mathbb{F}_r). If V has dimension d , then the number of elements in V equals r^d . For any $a \in \mathbb{F}_q$, we define $a + V$, the translate of V by a , as

$$a + V := \{a + v \mid v \in V\}.$$

Of course $a + V$ can also be seen as a coset of the subgroup $V \subset \mathbb{F}_q$ with addition as group operation. Any translate $a + V$ has r^d elements and moreover, it holds that two translates $a + V$ and $b + V$ are either disjoint or the same. This means that one can choose $n := r^e/r^d = r^{e-d}$ values of a , say a_1, \dots, a_n such that the sets $a_1 + V, \dots, a_n + V$ partition \mathbb{F}_q .

The next task is to describe for a given subspace V of dimension d , the $n := r^{d-e}$ polynomials with $a_1 + V, \dots, a_n + V$ as sets of roots. As a first step, we can just as before, construct an \mathbb{F}_r -linear map ϕ from F_q to itself, that can be described using a linearized polynomial of the form $p_V(X) = X^{r^d} + c_{d-1}X^{r^{d-1}} + \dots + c_1X^r + c_0X$. The linear map ϕ then simply sends x to $p_V(x)$ and has as image

$$W := \{p_V(x) \mid x \in \mathbb{F}_q\}.$$

A coset $a + V$ of V is then sent to the element $p_V(a)$ by ϕ . This means that any coset of V can be described as the set of roots of the polynomial $p_V(X) - p_V(a)$, that is to say of the form $p_V(X) - b$ with $b \in W$ (the image of the map ϕ). Combining this, we obtain that we can partition the elements of \mathbb{F}_q as sets of roots of polynomials of the form $p_V(X) - b$ with $b \in W$. Note that these polynomials still are very structured: just a constant term is added to the already very sparse polynomial $p_V(X)$. Note that $p_V(X) - p_V(a) = p_V(X - a)$, since $p_V(X)$ is a linearized polynomial. This makes it easy to confirm that indeed the set of roots of a polynomial of the form $p_V(X) - p_V(a)$ is just the coset $a + V$. The number of elements in W is easily calculated: since it is the image of the linear map ϕ and the dimension of the null space of ϕ is d (the dimension of V), the dimension of its image is $e - d$. This implies that W contains r^{e-d} elements. We collect some of this in the following proposition:

Proposition 2. *Let $q = r^e$ and let V be a linear subspace of \mathbb{F}_q of over the field \mathbb{F}_r of dimension d . Moreover denote by $p_V(x)$ be the linearized polynomial associated to V and define $W := \{p_V(x) \mid x \in \mathbb{F}_q\}$.*

Then for any $a \in \mathbb{F}_q$, the polynomial $p_V(x) - p_V(a)$ has as sets of roots exactly $a + V$. Moreover, the sets of roots of the polynomials $p_V(x) - b$ with $b \in W$ partition \mathbb{F}_q .

A possible description of a basis of W can be obtained in a fairly straightforward way. If $\{\beta_1, \dots, \beta_d\}$ is a basis of V , one can extend this to a basis of \mathbb{F}_q , say by adding the elements $\beta_{d+1}, \dots, \beta_e$. Then a basis of the image W of ϕ is simply given by the set $\{p_V(\beta_{d+1}), \dots, p_V(\beta_e)\}$ (note that $\phi(\beta_i) = p_V(\beta_i) = 0$ for $1 \leq i \leq d$). This means that the r^{e-d} polynomials whose roots partition \mathbb{F}_q are given by

$$p_V(X) + \sum_{i=d+1}^e a_i p_V(\beta_i), \quad \text{with } a_i \in \mathbb{F}_r.$$

The set of roots of a polynomial of this form is given by $\sum_{i=d+1}^e a_i \beta_i + V$. In the appendix, we give examples for $r^e = 2^{128}$ and $d = 31$ or $d = 61$.

5 Improved key Recovery Algorithm

Suppose that we have observed a polynomial hash collision for some forgery polynomial $p_V(X)$ of degree d , i.e. some observed message M and $M + p_V$ have the same image under h_H with the unknown authentication key H . This means that H must be among the roots of $p_V(X)$, and we can submit further verification queries using specially chosen forgery polynomials to recover the key.

5.1 An Explicit key Recovery Algorithm Using Twisted Polynomials

Being constructed in a twisted polynomial ring, our polynomials $p_V(X)$ are linearized polynomials, so that all roots are contained in a d -dimensional linear space $V \subset \mathbb{F}_2^n$. This enables an explicit and particularly efficient key recovery algorithm which recovers the key H by writing it as $H = \sum_{i=1}^d b_i \beta_i$ with respect to (w.r.t.) a basis $\mathcal{B} = \{\beta_1, \dots, \beta_d\}$ for V over \mathbb{F}_2 and determining its d binary coordinates w.r.t. \mathcal{B} one by one. Shortening the basis by the last element, we can test if $b_d = 0$ by using the forgery polynomial corresponding to $V' = \text{span}\{\beta_1, \dots, \beta_{d-1}\}$. If this query was not successful, we deduce $b_d = 1$. We then proceed recursively for the next bit.

Unless all $b_i = 0$, the search space will be restricted to an affine instead of a linear subspace at some point. It is easy to see, however, that the corresponding polynomial for $A = V + a$ with V a linear subspace, can always be determined as $p_A(X) = p_V(X - a) = p_V(X) - p_V(a)$ since the $p_V(X)$ are linearized polynomials.

The complexity of Algorithm 5.2 for a polynomial of degree d (corresponding to $|V| = 2^d$) is given by d verification queries and one invocation of the polynomial construction algorithm 5.1, which in turn takes $O(d^2)$ finite field operations. Note that typically, $d < 64$. The total length of all verification queries is limited by 2^{d+1} blocks. Since the polynomials $p_{U^{(i)}}(X)$ have at most $d+1$ nonzero coefficients, they are very sparse and only very few additions to M are required to compute the message $M + P_{U^{(i)}}$ for the forgery attempt.

We emphasize that this algorithm can be readily generalized to deal with input polynomials $p_A(X)$ having affine root spaces $A = V + a$ by operating with the corresponding linear space V and adding $p_{V^{(i)}}(a)$ to all verification queries. This especially allows to combine this algorithm with the key space covering strategy of Sect. 4.2.

In the context of authenticated encryption, M will typically correspond to ciphertexts instead of plaintexts, so also in this case, only calls to the verification oracle are required. It is also straightforward to adapt Algorithm 5.2 to cases where a polynomial hash collision cannot directly be observed, but instead propagates into some other property visible from ciphertext and tag. This is for example used in our attacks on the COBRA authenticated encryption scheme (see the full version [2] for details).

5.2 Comparison to Previous Work

The idea of using a binary search-type algorithm to recover authentication keys has previously been applied to various universal hashing-based MAC construc-

Algorithm 5.1. Construction of twisted polynomials

Input: basis $\mathcal{B} = \{\beta_1, \dots, \beta_d\}$ of $V \subset \mathbb{F}_2^n$

Output: polynomials $p_{V(i)}(X)$ having $\text{span}\{\beta_1, \dots, \beta_i\}$ as set of roots

- 1: Set $a_1 \leftarrow 1$
- 2: Set $a_i \leftarrow 0$ for $2 \leq i \leq d+1$
- 3: **for** $i = 1$ to d **do**
- 4: $v \leftarrow \sum_{k=1}^d a_k \beta_i^{2^k}$
- 5: $c_1 \leftarrow v \cdot a_1$
- 6: **for** $j = 2$ to $d+1$ **do**
- 7: $c_j \leftarrow a_{j-1}^2 + v \cdot a_j$
- 8: **end for**
- 9: $p_{V(i)} \leftarrow \sum_{k=1}^{d+1} c_k X^{2^{k-1}}$
- 10: **end for**
- 11: **return** polynomials $p_{V(1)}(X), \dots, p_{V(d)}(X)$

Algorithm 5.2. Key recovery using twisted polynomials

Input: message M , polynomial $p_V(X)$ s.t. $h_H(M) = h_H(M + P_V)$, basis $\mathcal{B} = \{\beta_1, \dots, \beta_d\}$ of d -dimensional linear subspace $V \subset \mathbb{F}_2^n$.

Output: authentication key H .

- 1: $b_i \leftarrow 0, 1 \leq i \leq d$
- 2: Call Alg. 5.1 on V , obtain $p_{V(1)}, \dots, p_{V(d)}$
- 3: **for** $i = d$ downto 1 **do**
- 4: Denote $U^{(i)} = \text{span}\{\beta_1, \dots, \beta_{i-1}\}$, so that $p_{U(i)} = p_{V(i-1)}$
- 5: $\alpha \leftarrow p_{U(i)}(\sum_{j=i}^d b_j \beta_j)$
- 6: **if** $h_H(M) = h_H(M + P_{U(i)} + \alpha)$ **then**
- 7: $b_i \leftarrow 0$
- 8: **else**
- 9: $b_i \leftarrow 1$
- 10: **end if**
- 11: **end for**
- 12: **return** key $H = \sum_{i=1}^d b_i \beta_i$

tions by Handschuh and Preneel [10]. Their attack algorithm however does not deal with the (important) questions of determining new polynomials having distinct roots from all previously used ones, and also requires the calculation and storage of the 2^d roots during the key search phase. Also, the required polynomials will not be sparse and require up to 2^d nonzero coefficients. By contrast, our algorithm leverages the twisted polynomial ring to explicitly construct sparse polynomials with exactly the necessary roots for restricting the search space in each iteration.

A different approach for binary-search type key recovery is given in Sect. 7.3 of [15], suggesting the use of forgery polynomial $q(X) = \prod_{H \in \mathbb{F}_2^n, H_n=0} (X - H)$ and then subsequently fixing more bits of H according to the results of the verification queries. While this is clearly optimal with respect to the number of queries (which is n), the resulting messages are up to 2^n blocks long, which typically exceeds the limits imposed by the specifications. Additionally, the polynomials will have almost no zero coefficients, which requires up to 2^{n+1} additions for the verification queries. By contrast, when combined with the keyspace covering strategy outlined in Sect. 4.2, our algorithm requires $2^{n/d} \cdot d$ queries, each of them being maximally 2^d blocks long. This not only allows staying within the specified limits, but also allows choosing any desired trade-off between the number and length of the queries. Our explicit polynomials also have a maximum of $d+1$ nonzero coefficients each, which limits the number of additions to $2^{n/d} \cdot (d+1)$.

6 Nonce-respecting Universal Forgeries for GCM

In this section, we describe two nonce-respecting universal forgery attacks against GCM [6] under weak keys. Before describing the attacks we describe the GCM authenticated encryption scheme and the GCM counter values generation procedure as defined in the NIST standard [6].

6.1 More Details on GCM

We recall the GCM ciphertext/tag generation:

$$T = E_k(J_0) \oplus h_H(C),$$

with T denoting the tag, with $M = M_1||M_2||\dots||M_l$ the plaintext and $C = C_1||C_2||\dots||C_l$ the ciphertext blocks produced using a block cipher E_k in counter mode, i.e. $C_i = E_K(J_{i-1}) \oplus M_i$. The J_i 's are successive counters with the initial J_0 generated from the nonce N ; furthermore $H = E_k(0)$ with k the secret key.

We now focus on the detailed generation of the counter values in GCM. We have

$$J_0 = \begin{cases} N||0^{31}||1 & \text{if } |N| = 96, \\ h_H(N||0^{s+64}||[N]_{64}) & \text{if } |N| \neq 96, \end{cases}$$

where $J_i = \text{inc}_{32}(J_{i-1})$, where $s = 128\lceil|N|/128\rceil - |N|$, $[X]_{64}$ is the 64-bit binary representation of X and $\text{inc}_{32}(X)$ increments the right-most 32 bits of the binary string X modulo 2^{32} ; the other left-most $|X| - 32$ bits remain unchanged.

6.2 Universal Forgery Attacks on GCM

Our universal forgery attacks are possible if the hash key H is weak. Therefore, our attack starts by detecting whether the hashing key H is weak or not using our forgery polynomial $q(X) = p_V(X)$ of degree 2^{31} explicitly described in Appendix A.1. In other words, we make a blind forgery for an observed ciphertext/tag pair $(C; T)$ by asking for the verification of the forged ciphertext $(C+Q); T$ where $Q = q_1||\dots||q_l$. Now if H is a weak key according to our forgery polynomial – is a root of $q(X) = p_V(X)$ – then the verification succeeds and the GCM scheme outputs a random plaintext.

Once we know that H is a weak-key, then we can recover it using Algorithm 5.2 over the roots of $q(X) = p_V(X)$ (see Appendix A.1) where at each query we can choose different nonces.

Now, the only hurdle for generating a nonce-respecting forgery is computing the value of $E_K(J_0)$ since we do not know the secret key K (we have only recovered $H = E_K(0)$). However, since GCM is using a counter mode encryption where the successive counter values J_i , are generated from the nonce, we can easily get the encryption of the counter values $E_K(J_i)$ by simply xorring the corresponding plaintext and ciphertext blocks (Note that in NIST GCM, the right-most 32 bits of the counter values are successive modulo 2^{32} as shown below). In the sequel, we show how to use the encryption of the counter values in order to construct universal forgeries.

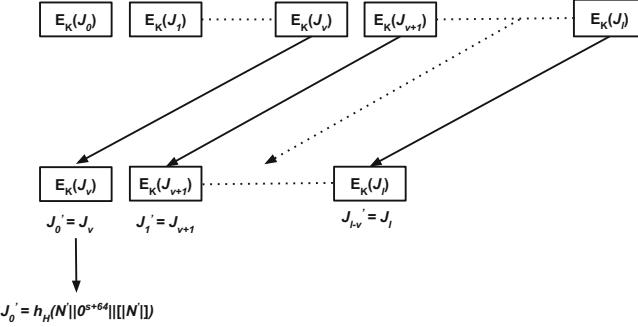


Fig. 1. Forgeries for GCM via sliding the counter encryptions

Slide Universal Forgeries Using Chosen Nonce N with $|N| \neq 96$ Suppose that we have observed an l -block plaintext/ciphertext with tag T , $M = M_1 || \dots || M_l$ and $C = C_1 || \dots || C_l$, where $C_i = M_i \oplus E_K(J_{i-1})$, $J_i = \text{inc}_{32}(J_{i-1})$ and $T = E_K(J_0) \oplus h_H(C)$. Our goal now is to generate a valid ciphertext/tag for a different message M' using a different chosen nonce N' where $|N'| \neq 96$.

As mentioned above, the counter mode of operation enables us to find the encryption of the counter values, $E_K(J_0), E_K(J_1), \dots, E_K(J_v), \dots, E_K(J_l)$. The idea of the attack is to slide these encrypted counter values v positions to the left in order to re-use the $(l - v)$ encrypted counter values $E_K(J_v), \dots, E_K(J_l)$ to generate valid ciphertext/tag for any new message M' with a new chosen nonce N' that gives us an initial counter value $J'_0 = J_v$. This will enable us to make slide universal forgeries for an $(l - v)$ -block message. See Fig. 1.

One can see that using J_v , $v > 0$, it is possible to choose a nonce N' that gives $J'_0 = J_v$ by solving the following equation for N'

$$J'_0 = J_v = h_H(N' || 0^{s+64} || [N'])_64$$

Note that when $|N'| = 128$ (i.e. $s = 0$), we have only one solution for N' and more than one solution for $|N'| > 128$. However, when $|N'| < 128$ we might have no solution. Therefore we assume that $|N'| \geq 128$.

Once we find the nonce N' that yields $J'_0 = J_v$, then one can see that we have the following ‘slid’ identities:

$$E_K(J'_0) = E_K(J_v), E_K(J'_1) = E_K(J_{v+1}), \dots, E_K(J'_{l-v}) = E_K(J_l)$$

Consequently, we are able to compute $C'_i = M'_i \oplus E_K(J'_{i-1})$ for $1 \leq i \leq l - v$ and $T' = E_K(J'_0) \oplus h_H(C')$. Thus observing the encryption of an l -block message and setting $J'_0 = J_v$ as shown above enable us to generate a valid ciphertext/tag (C'/T') for an $(l - v)$ -block message M' under the nonce-respecting setting.

Universal Forgeries Using Arbitrary Nonces N with $|N| = 96$. Assume that we are using a GCM implementation that supports variable nonce lengths. For example, the implementation of GCM in the latest version of OpenSSL

[17, 18] makes the choice of the nonce length optional, i.e. one can use different nonce sizes under the same secret key. Now, suppose that using such a GCM oracle with the secret key K , we need to find the ciphertext/tag of a message $M = M_1 \parallel \dots \parallel M_l$ with a nonce N where $|N| = 96$, so $J_0 = N \parallel 0^{31} \parallel 1$. In order to generate the ciphertext/tag we need to find $E_K(J_i)$ where $J_i = \text{inc}_{32}(J_{i-1})$. We do not know the secret key K . However, since we know the secret hash key H , we can solve for N' the following equation

$$J_0 = h_H(N' \parallel 0^{s+64} \parallel [|N'|]_{64}) \quad \text{where} \quad |N'| \neq 96$$

Note that we assume that $|N'| \geq 128$ as otherwise we might not get a solution. After finding N' , we can query the same GCM oracle (that has been queried for encrypting M with the nonce N where $|N| = 96$) with a new nonce N' that has a different size $|N'| \geq 128$ ² for the encryption of some plaintext $M = M'_1 \parallel \dots \parallel M'_l$. Now, $|N'| \neq 96$ means that the initial counter value $J'_0 = h_H(N' \parallel 0^{s+64} \parallel [|N'|]_{64}) = J_0$. Therefore, from the corresponding ciphertext blocks C'_1, \dots, C'_l , we find $E_K(J_i) = E_K(J'_i) = M'_i \oplus C'_i$. Consequently the corresponding i th ciphertext block of M_i is $C_i = E_K(J'_i) \oplus M_i$ and the corresponding tag is $T = E_K(J'_0) \oplus h_H(C)$. It is worthy to note, that this interaction possibility between two different nonce lengths on GCM had been listed in [19] as one of the undesirable characteristics of GCM. Fig. 2 demonstrates the interaction attack.

7 Analysis of POET

In this section, we present a detailed weak key analysis of the online authentication cipher POET when instantiated with Galois-Field multiplication. More specifically, we create universal forgery attacks once we recover the hashing weak key. Before this we give a brief description of POET.

7.1 Description of POET

A schematic description of POET [3] is given in Fig. 3a. Five keys $L, K, L_{\text{top}}, L_{\text{bot}}$ and L_T are derived from a user key as encryptions of the constants $1, \dots, 5$. K denotes the block cipher key, L is used as the mask in the AD processing, and L_T is used as a mask for computing the tag. Associated data (AD) and the nonce are processed using the secret value L in a PMAC-like fashion (see [3] for details) to produce a value τ which is then used as the initial chaining value for both top and bottom mask layers, as well as for generating the authentication tag T . The “header” H encompasses the associated data (if present) and includes the nonce

² Two of the test vectors (Test Case 3 and Test Case 6, see the full version [2]) for the GCM implementation in the latest release of OpenSSL share the same secret key (and therefore the same hash key) but they use different nonce sizes, Test Case 3 uses a nonce with length 96 while Test Case 6 uses a nonce with length 480 [18]. This suggests that it is conceivable to have different IV sizes under the same secret key.

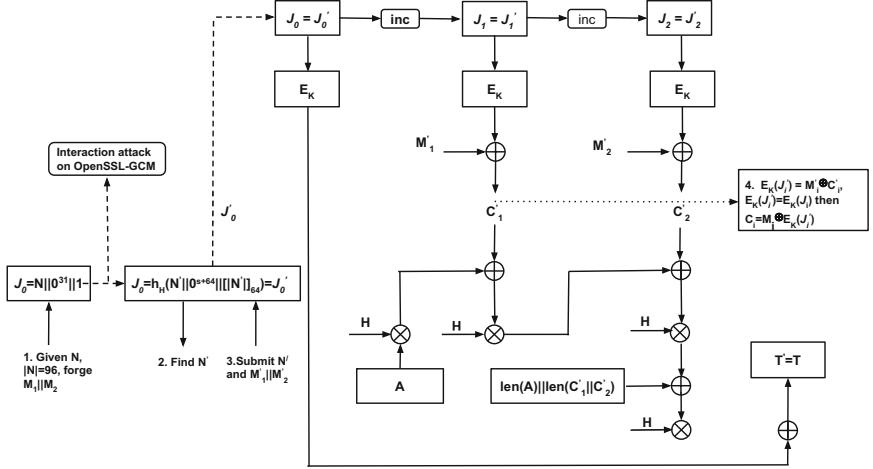


Fig. 2. Forgeries for GCM via cross-nonce interaction

in its last block. S denotes the encryption of the bit length of the message M , i.e. $S = E_K(|M|)$. The inputs and outputs of the i -th block cipher call during message processing are denoted by X_i and Y_i , respectively.

One of the variants of POET instantiates the functions F_t and F_b by $F_t(x) = L_{\text{top}} \cdot x$ and $F_b(x) = L_{\text{bot}} \cdot x$, with the multiplication taken in \mathbb{F}_2^{128} . This is also the variant that we consider in this paper. The top AXU hash chain then corresponds to the evaluation of a polynomial hash in \mathbb{F}_2^{128} :

$$g_t(X) = \tau L_{\text{top}}^m + \sum_{i=1}^m X_i L_{\text{top}}^{m-i},$$

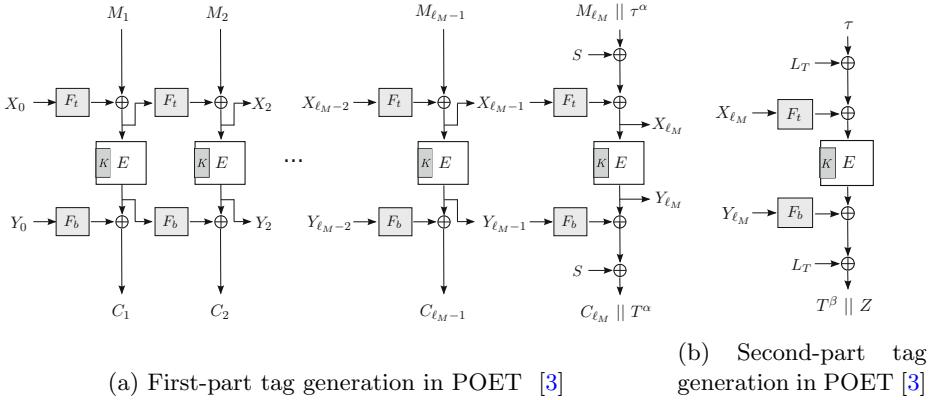
with g_t being evaluated at $X = M_1, \dots, M_{m-1}, M_m \oplus S$.

For integral messages (i.e., with a length a multiple of the block size), the authentication tag T then generated as $T = T^\beta$ with empty Z , as shown in Fig. 3b. Otherwise, the tag T is the concatenation of the two parts T^α and T^β , see Fig. 3a and 3b.

7.2 Universal Weak-key Forgeries for POET

We start by the following observations.

Observation 1 (Collisions in g_t imply tag collisions). Let $M = M_1, \dots, M_m$ and $M' = M'_1, \dots, M'_m$ be two distinct messages of m blocks length such that $g_t(M) = g_t(M')$ or $g_t(M_1, \dots, M_\ell) = g_t(M'_1, \dots, M'_\ell)$ with $\ell < m$ and $M_i = M'_i$ for $i > \ell$. This implies a collision on POET's internal state X_i, Y_i for $i = m$ or $i = \ell$ respectively, and therefore equal tags for M and M' .

**Fig. 3.** Schematic description of POET

We note that such a collision also allows the recovery of L_{top} by Algorithm 5.2.

Observation 2 (Knowing L_{top} implies knowing L_{bot}). Once the first hash key L_{top} is known, the second hash key L_{bot} can be determined with only two 2-block queries: Choose arbitrary M_1, M_2, ∇_1 with $\nabla_1 \neq 0$ and obtain the encryptions of the two 2-block messages M_1, M_2 and M'_1, M'_2 with $M'_1 = M_1 \oplus \nabla_1, M'_2 = M_2 \oplus \nabla_1 \cdot L_{\text{top}}$. Denote $\Delta_i = C_i \oplus C'_i$. Then we have the relation $\Delta_1 \cdot L_{\text{bot}} = \Delta_2$, so $L_{\text{bot}} = \Delta_1^{-1} \cdot \Delta_2$.

It is worth noting that this procedure works for arbitrary L_{bot} , and is in particular not limited to L_{bot} being another root of the polynomial q .

A Generic Forgery. In the setting of [15], consider an arbitrarily chosen polynomial $q(X) = \sum_{i=1}^{m-1} q_i X^i = p_V(X)$ of degree $m - 1$ and some message $M = M_1 \parallel \cdots \parallel M_{m-1} \parallel M_m$. Write $Q = q_1 \parallel \cdots \parallel q_{m-1}$ and define $M' \stackrel{\text{def}}{=} M + Q$ with Q zero-padded as necessary. For a constant nonce (1-block header) H , denote ciphertext and tag corresponding to M by $C = C_1, \dots, C_m$ and T , and ciphertext and tag corresponding to $M' = M + Q$ by $C' = C'_1, \dots, C'_m$ and T' , respectively.

If some root of q is used as the key L_{top} , we have a collision between M and $M' = M + Q$ in the polynomial hash evaluation after $m - 1$ blocks:

$$\tau L_{\text{top}}^m + \sum_{i=1}^{m-1} M_i L_{\text{top}}^{m-i} = \tau' L_{\text{top}}^m + \sum_{i=1}^{m-1} M'_i L_{\text{top}}^{m-i}$$

This implies $X_{m-1} = X'_{m-1}$ and therefore $Y_{m-1} = Y'_{m-1}$. Since the messages are of equal length, $S = S'$ and we also have a collision in X_m and Y_m . It follows that $C_m = C'_m$. Furthermore, since $\tau = \tau'$, the tag T is colliding as well. Since then M and $M + Q$ have the same tag, $M + Q$ is a valid forgery whenever some

root of $q(X) = p_V(X)$ is used as L_{top} . Note that both M and the forged message will be m blocks long.

Using the class of weak keys represented by the roots of the forgery polynomial $q(X) = p_V(X)$ explicitly described in Appendix A and Appendix A.2, we discuss the implication of having one such key as the universal hash key L_{top} . Since POET allows nonce-reuse, we consider nonce-repeating adversaries, i.e. for our purposes, the nonce will be fixed to some constant value for all encryption and verification queries. However, once we recovered τ , we will be able to recover the secret value L and consequently we can make forgeries without nonce-reuse.

More specifically, we show that weak keys enable universal forgeries for POET under the condition that the order of the weak key is smaller than the maximal message length in blocks. For obtaining universal forgeries, we first use the polynomial hash collision described above to recover the weak keys L_{top} and L_{bot} , and then recover τ , which is equal to the initial states X_0 and Y_0 , under the weak key assumption.

Recovering τ . Suppose that we have recovered the weak keys L_{top} and L_{bot} . Now our goal is to recover the secret $X_0 = Y_0 = \tau$. We know that $X_i = \tau L_{\text{top}}^i + M_1 L_{\text{top}}^{i-1} + M_2 L_{\text{top}}^{i-2} + \dots + M_i$ and $X_{i+j} = \tau L_{\text{top}}^{i+j} + M_1 L_{\text{top}}^{i+j-1} + M_2 L_{\text{top}}^{i+j-2} + \dots + M_{i+j}$.

Now if L_{top} has order j , i.e. $L_{\text{top}}^j = \text{Identity}$, then we get $X_i = X_{i+j}$ by constructing M_{i+1}, \dots, M_{i+j} such that $M_{i+1}L_{\text{top}}^{j-1} + M_{i+2}L_{\text{top}}^{j-2} + \dots + M_{i+j} = 0$. The easiest choice is to set $M_{i+1} = M_{i+2} = \dots = M_{i+j} = 0$. This gives us $Y_i = Y_{i+j}$. Now equating the following two equations and assuming that $L_{\text{bot}}^j \neq \text{Identity}$, $Y_i = \tau L_{\text{bot}}^i + C_1 L_{\text{bot}}^{i-1} + C_2 L_{\text{bot}}^{i-2} + \dots + C_i$ and $Y_{i+j} = \tau L_{\text{bot}}^{i+j} + C_1 L_{\text{bot}}^{i+j-1} + C_2 L_{\text{bot}}^{i+j-2} + \dots + C_{i+j}$. We get

$$\begin{aligned}\tau &= (C_1 L_{\text{bot}}^{i-1} + C_2 L_{\text{bot}}^{i-2} + \dots + C_i \\ &\quad + C_1 L_{\text{bot}}^{i+j-1} + C_2 L_{\text{bot}}^{i+j-2} + \dots + C_{i+j}) \cdot (L_{\text{bot}}^i + L_{\text{bot}}^{i+j})^{-1},\end{aligned}$$

which means that we now know the initial values of the cipher state.

Querying POET’s Block Cipher E_K . One can see from Fig. 3a that once we know L_{top} , L_{bot} and τ , we can directly query POET’s internal block cipher without knowing its secret key K . internal block cipher, i.e. we want to compute $E_K(x)$. Now from Fig. 3a, we see that the following equation holds: $E_K(\tau L_{\text{top}} \oplus M_1) = C_1 \oplus \tau L_{\text{bot}}$, therefore $E_K(x) = C_1 \oplus \tau L_{\text{bot}}$. If M_1 was the last message block, however, we would need the encryption $S = E_K(|M|)$. Therefore we have to extend the auxiliary message for the block cipher queries by one block, yielding the following:

Observation 3 (Querying POET’s block cipher). *Knowing L_{top} , L_{bot} and τ enables us to query POET’s internal block cipher without the knowledge of its secret key K . To compute $E_K(x)$ for arbitrary x , we form a two-block auxiliary message $M'_1 = (x \oplus \tau L_{\text{top}}, M'_2)$ for arbitrary M'_2 and obtain its POET encryption*

as C'_1, C'_2 . Computing $E_K(x) := C'_1 \oplus \tau L_{\text{bot}}$ then yields the required block cipher output.

This means that we can produce valid ciphertext blocks C_1, \dots, C_{ℓ_M} and (if necessary) partial tags T^α for any desired messages, by simply following the POET encryption algorithm using the knowledge of $L_{\text{top}}, L_{\text{bot}}, \tau$ and querying POET with the appropriate auxiliary messages whenever we need to execute an encryption E_K . Note that this also includes the computation of $S = E_K(|M|)$.

Generating the Final Tag. In order to generate the second part of the tag T^β (see Fig. 3b), which is the full tag T for integral messages, we use the following procedure.

We know the value of X_{ℓ_M} for our target message M from the computation of C_{ℓ_M} . If we query the tag for an auxiliary message M' with the same $X'_{\ell_{M'}}$, the tag for M' will be the valid tag for M as well, since having $X'_{\ell_{M'}} = X_{\ell_M}$ means that $Y'_{\ell_{M'}} = Y_{\ell_M}$ and consequently $T^{\beta'} = T^\beta$.

Therefore, we construct an auxiliary one-block message $M' = (X_{\ell_M} \oplus E_K(|M'|) \oplus \tau L_{\text{top}})$ and obtain its tag as T' (computing the encryption of the one-block message length by querying E_K as above). By construction $X'_1 = X_{\ell_M}$, so T' is the correct tag for our target message M as well.

By this, we have computed valid ciphertext blocks and tag for an arbitrary message M by only querying some one- or two-block auxiliary messages. This constitutes a universal forgery.

We finish by noting that in case a one- or two-block universal forgery is requested, we artificially extend our auxiliary messages in either the final tag generation (for one-block targets) or the block cipher queries (for two-block messages) with one arbitrary block to avoid having queried the target message as one of our auxiliary message queries.

7.3 Further Forgery Strategies

Since the universal forgery of the previous section relies on having a weak key L_{top} with an order smaller than the maximum message length for recovering τ , we describe two further forgery strategies that are valid for any weak key, regardless of its order. We also show how the knowledge of τ enables us to recover the secret value L . This will enable us to make universal forgeries on POET within the nonce-respecting adversary model. In other words, recovering the secret value L means that we will be able to process the header (associated data and nonce) and generate a new τ and consequently have a total control over the POET scheme. Due to the space limitation, all these further forgery attacks are given in the full version [2] of this paper.

8 Conclusion

Polynomial hashing is used in a large number of MAC and AE schemes to derive authentication tags, including the widely deployed and standardized GCM, and

recent nonce misuse-resistant proposals such as POET, Julius, and COBRA. While a substantial number of works has pointed out weaknesses stemming from its algebraic structure [10, 15, 20], a crucial part of the proposed attacks, the construction of appropriate forgery polynomials, had not been satisfactorily addressed.

In this paper, we deal with this open problem of polynomial construction and selection in forgery and key recovery attacks on such schemes. We describe explicit constructions of such forgery polynomials with controlled sets of roots that have the additional advantage of being very sparse. Based upon this, we propose two strategies to achieve complete disjoint coverage of the key space by means of such polynomials, again in an explicit and efficiently computable construction. We also saw that this yields an improved strategy for key recovery in such attacks.

We then apply our framework to GCM in the weak-key model and describe, to the best of our knowledge, the first universal forgeries without nonce reuse. We also describe such universal forgeries for the recent AE schemes POET, Julius, and COBRA.

A Appendix: Forgery Polynomial Suggestions for GCM and POET

In this appendix we give some examples of polynomials whose roots form a linear subspace V_d of $\mathbb{F}_{2^{128}}$ of dimension d for $d = 31$ and $d = 61$. As vector space V_d we have chosen the space spanned by the elements $1, \gamma, \dots, \gamma^{d-1}$, with γ a primitive element of $\mathbb{F}_{2^{128}}$ satisfying $\gamma^{128} = \gamma^7 + \gamma^2 + \gamma + 1$. The calculated polynomial will have the form $c_{d+1}X^{2^d} + c_dX^{2^{d-1}} + \dots + c_1X^{2^0}$ and it is sufficient to simply state the coefficients c_i , which can be expressed in the form a^{e_i} with $0 \leq e_i \leq 2^{128} - 2$. To save space we only list the exponents e_i for each polynomial in the following tables.

A.1 Forgery Polynomial with Degree 2^{31} for Attacking GCM

For $d = 31$, one obtains the following coefficients:

Table 1. The table shows the coefficients of the forgery polynomial $q(X) = p_V(X)$ for attacking GCM

i	e_i
1	57661364709898787973942162593394430677
2	88640585123887860771282360281650849369
3	2284676975914793351730666079059941262
4	60870920642211311860125058878376239967
5	69981393859668264373786090851403919597
6	25545984420946355543584538974500206397
7	263576500668765237830541241929740306586
8	37167015149451472008716003077656492621
9	58043277378748107723324135119415484405
10	321767455835401530567257366419614234023
11	4503388854150737621429712394846444657
12	258425985086309803122357832308421510564
13	105831989526232747717837668269825340779
14	267464360177071786386745024557199320756
15	280644372754658909872880662034708629284
16	1050003268562505697615431403289357708609
17	4582581835946061154228322536890182857
18	82845961308169259876601267127459416989
19	44217989936194208472522353821220861115
20	69062943960552309089842983129403174217
21	26846201940483608935933493976220681511
22	30001648942113240212113555293749765514
23	669737854382487997736546203881056449
24	127958856468256956044189872000451203235
25	2771622338678239965835219683143318848400
26	134662498954166373112542807113066342554
27	219278415175240762588240883266619436470
28	216197476010311230105259534730909158682
29	281783005767613667130380044536264251829
30	181483131639777656403198412151415404929
31	3838486687611426333051602240884584792
32	0

Table 2. The table shows the coefficients of the forgery polynomial $q(X) = p_V(X)$ for attacking POET

i	e_i
1	20526963135026773119771529419991247327
2	264546851691026540251722618719245777504
3	7927973230583347490289364797721594921
4	3257125558590854229153756018169632351
5	281140838798434203589324885473621429913
6	271147943451442547572675283203493235775
7	33255255082373325202932488407731432338
8	6718016882907633170860567569329895273
9	2558890659818386790319621991012154253
10	49457687721601463712640189217755474230
11	3115900544256973027703075522863616807
12	2279845104054619648939249132690066393
13	3246609530451183283558900161997992161
14	10137005974578928512751937970042717674145948076404968
15	335840777837142047556550075244373419708
16	31458890659818386790319621991012154253
17	339477818976914242962960654286547702007
18	267056244491330957618685443721979120206
19	115274327651619347046091793992432007152
20	3096064718383261068453469483105904888
21	31472831963470533804935496732997263
22	191332595597193424626322329032056378009
23	18955391343109255614514163550670075672
24	224617322052671248319257827067474740867
25	6304123030678803297311114533307051562
26	2215766062721523541533073975040337239
27	2917999035400626289220245045188573741192
28	290489624437950764499707232619770186293
29	263754726506046639985479240606003777000
30	45160807436167307990689150792052670707
31	33630881905996630925327701622950425950
32	109604555581389038896555752982244394616
33	119482829110451460647031381779266776526
34	1652978586103801312499481664434468967
35	155444340258770748055544634836807134293
36	86982184438730045821274025831061961430
37	104870645496065737272877350967826010844
38	56281281579002318337037919356127105369
39	1000685189828379284718705774049983141
40	93687920075554812358890244898088345449
41	69826727900303432248401753658262533506
42	246360754285298743574294101515912517720
43	895678936019042717674145948076404968
44	3376817267807031517220235608097231854
45	21031754700430237276427434844069094761
46	15857432113301045534802861165087620178
47	291559826228649927512447763293001897434
48	1563512433124423160769052717791457746
49	196562458398036090488379086660199368109
50	308779188958300135859037769338975723488
51	311961723579011854596575128443762996895
52	153505384649696850323974564044760550270
53	136880473479137548264080346617303001989
54	325361660912502344542873376867973189476
55	75648626101374794093175916332043285057
56	12290403576559817931510431150449672627
57	240654906561678387738109953233510366
58	71774746460316463981542974558280761865
59	3188339703714313727629357160120999244730
60	176351990917361872511208705771673004140
61	227372417807158122619428517134408021585
62	0

A.2 Forgery Polynomial with Degree 2^{61} for Attacking POET

Similarly for $d = 61$ one obtains the following coefficients:

Let us denote the found polynomials by $p_d(X)$ (with $d = 31$ or $d = 61$). From $p_d(X)$, we can obtain a family of 2^{128-d} polynomials whose root sets partition

\mathbb{F}_2^{128} . The polynomials have the form $p_d(X) + b$, with $b \in W_d := \{p_d(a) \mid a \in \mathbb{F}_2^{128}\}$. Since in the above examples V_d has basis $\{1, \gamma, \dots, \gamma^{d-1}\}$ A basis of W_d is given by $\{p_d(\gamma^i) \mid d \leq i \leq 127\}$, making it straightforward to describe all possibilities for b .

References

1. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, March 2014. <http://competitions.cr.yp.to/caesar.html>
2. Abdelraheem, M.A., Beelen, P., Bogdanov, A., Tischhauser, E.: Twisted Polynomials and Forgery Attacks on GCM. IACR ePrint Archive (2015)
3. Abed, F., Fluhrer, S., Foley, J., Forler, C., List, E., Lucks, S., McGrew, D., Wenzel, J.: The POET Family of On-Line Authenticated Encryption Schemes. Submission to the CAESAR competition, March 2014
4. Andreeva, E., Luykx, A., Mennink, B., Yasuda, K.: COBRA: a parallelizable authenticated online cipher without block cipher inverse. In: Cid, C., Rechberger, C. (eds.) Fast Software Encryption, FSE 2014. LNCS, p. 24. Springer (2014) (to appear)
5. Bahack, L.: Julius: Secure Mode of Operation for Authenticated Encryption Based on ECB and Finite Field Multiplications. Submission to the CAESAR competition, March 2014. <http://competitions.cr.yp.to/round1/juliusv10.pdf>
6. Doworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007. <http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf>
7. Ferguson, N.: Authentication weaknesses in GCM. Comments submitted to NIST Modes of Operation Process (2005)
8. Goss, D.: Basic structures of function field arithmetic. *Ergebnisse der Mathematik und ihrer Grenzgebiete (3) [Results in Mathematics and Related Areas (3)]*, vol. 35. Springer, Berlin (1996)
9. Guo, J., Jean, J., Peyrin, T., Lei, W.: Breaking POET Authentication with a Single Query. Cryptology ePrint Archive, Report 2014/197 (2014) <http://eprint.iacr.org/>
10. Handschuh, H., Preneel, B.: Key-recovery attacks on universal hash function based MAC algorithms. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 144–161. Springer, Heidelberg (2008)
11. Joux, A.: Authentication Failures in NIST version of GCM. Comments submitted to NIST Modes of Operation Process (2006)
12. McGrew, D., Fluhrer, S., Lucks, S., Forler, C., Wenzel, J., Abed, F., List, E.: Pipelineable on-line encryption. In: Cid, C., Rechberger, C. (eds.) Fast Software Encryption, FSE 2014. LNCS, p. 24. Springer (2014) (to appear)
13. McGrew, D., Viega, J.: The galois/counter mode of operation (gcm). Submission to NIST (2004). <http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf>
14. Nandi, M.: Forging attacks on two authenticated encryptions cobra and poet. Cryptology ePrint Archive, Report 2014/363 (2014). <https://eprint.iacr.org/2014/363>
15. Procter, G., Cid, C.: On weak keys and forgery attacks against polynomial-based MAC schemes. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 287–304. Springer, Heidelberg (2014)
16. Procter, G., Cid, C.: On weak keys and forgery attacks against polynomial-based mac schemes. Cryptology ePrint Archive, Report 2013/144 (2013). <http://eprint.iacr.org/>

17. OpenSSL Project. <https://www.openssl.org/>
18. OpenSSL Project. GCM Implementation: crypto/modes/gcm128.c. <https://www.openssl.org/source/> (latest release: April 7, 2014) (openssl-1.0.1g)
19. Rogaway, P.: Evaluation of some blockcipher modes of operation. Evaluation carried out for the Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan (2011)
20. Saarinen, M.-J.O.: Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 216–225. Springer, Heidelberg (2012)
21. Wegman, M.N., Carter, J.L.: New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences **22**(3), 265–279 (1981)

Lattices

Quadratic Time, Linear Space Algorithms for Gram-Schmidt Orthogonalization and Gaussian Sampling in Structured Lattices

Vadim Lyubashevsky^{1,2(✉)} and Thomas Prest^{2,3}

¹ INRIA, Paris, France

² École Normale Supérieure, Paris, France

{vadim.lyubashevsky,thomas.prest}@ens.fr

³ Thales Communications and Security, Gennevilliers, France

Abstract. A procedure for sampling lattice vectors is at the heart of many lattice constructions, and the algorithm of Klein (SODA 2000) and Gentry, Peikert, Vaikuntanathan (STOC 2008) is currently the one that produces the shortest vectors. But due to the fact that its most time-efficient (quadratic-time) variant requires the storage of the Gram-Schmidt basis, the asymptotic space requirements of this algorithm are the same for general and ideal lattices. The main result of the current work is a series of algorithms that ultimately lead to a sampling procedure producing the same outputs as the Klein/GPV one, but requiring only linear-storage when working on lattices used in ideal-lattice cryptography. The reduced storage directly leads to a reduction in key-sizes by a factor of $\Omega(d)$, and makes cryptographic constructions requiring lattice sampling much more suitable for practical applications.

At the core of our improvements is a new, faster algorithm for computing the Gram-Schmidt orthogonalization of a set of vectors that are related via a *linear isometry*. In particular, for a linear isometry $r : \mathbb{R}^d \rightarrow \mathbb{R}^d$ which is computable in time $O(d)$ and a d -dimensional vector \mathbf{b} , our algorithm for computing the orthogonalization of $(\mathbf{b}, r(\mathbf{b}), r^2(\mathbf{b}), \dots, r^{d-1}(\mathbf{b}))$ uses $O(d^2)$ floating point operations. This is in contrast to $O(d^3)$ such operations that are required by the standard Gram-Schmidt algorithm. This improvement is directly applicable to bases that appear in ideal-lattice cryptography because those bases exhibit such “isometric structure”. The above-mentioned algorithm improves on a previous one of Gama, Howgrave-Graham, Nguyen (EUROCRYPT 2006) which used different techniques to achieve only a constant-factor speed-up for similar lattice bases. Interestingly, our present ideas can be combined with those from Gama et al. to achieve an even larger practical speed-up.

We next show how this new Gram-Schmidt algorithm can be applied towards lattice sampling in quadratic time using only linear space. The main idea is that rather than pre-computing and storing the Gram-Schmidt vectors, one can compute them “on-the-fly” while running the sampling algorithm. We also rigorously analyze the required arithmetic precision

V. Lyubashevsky—This research was partially supported by the ANR JCJC grant “CLE”.

necessary for achieving negligible statistical distance between the outputs of our sampling algorithm and the desired Gaussian distribution. The results of our experiments involving NTRU lattices show that the practical performance improvements of our algorithms are as predicted in theory.

1 Introduction

Sampling lattice points is one of the fundamental procedures in lattice cryptography. It is used in hash-and-sign signatures [GPV08], (hierarchical) identity-based encryption schemes [GPV08, CHKP10, ABB10], standard-model signatures [ABB10, Boy10], attribute-based encryption [BGG+14], and many other constructions. Being able to output shorter vectors leads to more secure schemes, and the algorithm that produces the currently-shortest samples is the randomized version of Babai’s nearest-plane algorithm [Bab86] due to Klein [Kle00] and Gentry, Peikert, Vaikuntanathan [GPV08].

The main inefficiency of cryptography based on general lattices is that the key size is usually (at least) quadratic in the security parameter, which is related to the fact that a d -dimensional lattice is generated by d vectors. For security, the lattice dimension is usually taken to be on the order of 512, and this results in keys that are larger than one megabyte in size and unsuitable for most real-world applications. For this reason, all practical implementations of lattice schemes (e.g. [HPS98, LMPR08, LPR13a, DDLL13, DLP14]) rely on the hardness of problems involving polynomial rings [PR06, LM06, LPR13a], in which lattices can be represented by a few polynomials. Due to the fact that solving certain average-case problems over polynomial rings was shown to be as hard as solving worst-case problems in *ideal lattices* [STX09, LPR13a], building cryptographic systems using polynomials is often referred to as ideal lattice cryptography.

During its execution, the Klein/GPV algorithm is implicitly computing the Gram-Schmidt orthogonalization of the input basis.¹ Since the Gram-Schmidt procedure requires $\Theta(d^3)$ operations, the Klein/GPV sampler also requires at least this much time. For improving the time-complexity, one can pre-compute and store the Gram-Schmidt basis, which results in a sampling procedure that uses only $\Theta(d^2)$ operations. The Gram-Schmidt basis, however, requires the storage of $\Theta(d^2)$ elements, and so the key size is, as mentioned above, unacceptably large. One may hope that, again, using polynomials results in a decrease of the required storage. In this case, unfortunately, such rings do not help. The Gram-Schmidt orthogonalization procedure completely destroys the nice structure of polynomial lattices. So while a polynomial lattice basis can be represented by a few vectors, its Gram-Schmidt basis will have d vectors. Thus the $\Theta(d^2)$ -operation Klein/GPV algorithm requires as much storage when using polynomial lattices as when using general lattices, and is equally unsuitable for practical purposes. Therefore the only realistic solution is to not store the pre-processed

¹ The Gram-Schmidt procedure produces a mutually-orthogonal set of vectors $(\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_d)$ that span the same inner-product space as the input vectors $(\mathbf{b}_1, \dots, \mathbf{b}_d)$.

Gram-Schmidt basis, which would then allow for the polynomial lattice algorithm to be linear-space (since only the original, compact-representation basis needs to be stored), but require at least $\Omega(d^3)$ time due to the fact that the Gram-Schmidt basis will need to be computed.

1.1 Our Results

Our main result is an algorithm that computes the Gram-Schmidt basis of certain algebraic lattices using $\Theta(d^2)$, instead of $\Theta(d^3)$, arithmetic operations. We then show how this new procedure can be combined with the Klein/GPV sampler to achieve a “best-of-both-worlds” result – a sampling algorithm that requires $\Theta(d^2)$ operations, which does not require storing a pre-processed Gram-Schmidt basis. In ideal lattice cryptography, this implies being able to have keys that consist of just the compact algebraic basis requiring only linear storage. Not pre-computing the Gram-Schmidt basis of course necessarily slows down our sampling algorithm versus the one where this basis is already stored in memory. But our new orthogonalization algorithm is rather efficient, and the slowdown in some practical applications is by less than a factor of 4 (see the performance table in Section 9.1). In case of amortization (i.e. sampling more than one vector at a time), the running time of our new algorithm becomes essentially the same as that of the one requiring the storage of the orthogonalized basis. As a side note, since the Klein/GPV algorithm is just a randomized version of the classic Babai nearest plane algorithm [Bab86], all our improvements apply to the latter as well.

While analyzing the running-time of lattice algorithms, it is very important to not only consider the number of arithmetic operations, but also the arithmetic precision required for the algorithms to be stable. The run-time of algorithms that are not numerically stable may suffer due to the high precision required during their execution. A second important issue is understanding how the precision affects the statistical closeness of the distribution of the outputted vectors versus the desired distribution. We rigorously show that in order to have statistical closeness between the output distribution and the desired discrete Gaussian one be at most $2^{-\lambda}$, the required precision of the Gram-Schmidt basis needs to be a little more (in practice, less than a hundred bits) than λ . We then experimentally show that our new Gram-Schmidt procedure is rather numerically stable, and the intermediate storage is not much more than the final required precision.² A third issue that also needs to be considered in practice is the space requirements of the algorithm during run-time. While the stored basis is very short, it could be that the intermediate computations require much larger storage (e.g.

² The reason that we only do experimental analysis of this second part, rather than a rigorous theoretical one, is because the precision required in practice will be much less than in theory due to the fact that theoretically, there could exist bases on which the Gram-Schmidt procedure is rather unstable. When using this procedure in cryptographic schemes, we can always test our basis for numerical stability by comparing the output of the exact Gram-Schmidt algorithm versus the “approximate” Gram-Schmidt algorithm that will be actually used.

if the intermediate computation requires storing the entire Gram-Schmidt basis to a high precision). Our sampling algorithm, however, is rather efficient in this regard because it only requires storing one Gram-Schmidt *vector* at a time. The storage requirement during run-time is therefore less than 64KB for typical dimensions used in cryptographic applications.

Isometries and Ideal Lattices. Interestingly, our improved orthogonalization algorithm for polynomial lattices does not have much to do with their algebraic structure, but rather relies on their implicit geometric properties. The types of bases whose Gram-Schmidt orthogonalization we speed up are those that consist of a set of vectors that are related via a *linear isometry*. In particular, if H is a d -dimensional Hermitian inner-product space and $r : H \rightarrow H$ is a linear map that preserves the norm and is computable using $\mathcal{O}(d)$ operations, then we show (both theoretically and via implementations) that orthogonalizing a set of vectors $\{\mathbf{b}, r(\mathbf{b}), r^2(\mathbf{b}), \dots, r^{d-1}(\mathbf{b})\}$ can be done using $\Theta(d^2)$ floating point operations.

We now explain the connection between isometries and ideal lattices. Consider the cyclotomic number field, with the usual polynomial addition and multiplication operations, $F = \mathbb{Q}[X]/\langle \Phi_m(X) \rangle$ where $\Phi_m(X)$ is the m^{th} cyclotomic polynomial (and so it has degree $\phi(m)$). Elements in F can be represented via a *canonical embedding*³ into $\mathbb{C}^{\phi(m)}$, and in that case F becomes isomorphic, as an inner product space, to $\mathbb{R}^{\phi(m)}$ where the inner product $\langle \mathbf{a}, \mathbf{b} \rangle$ of $\mathbf{a}, \mathbf{b} \in F$ is defined as

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{1 \leq i \leq m, \gcd(i, m)=1} \overline{\mathbf{a}(\zeta_m^i)} \cdot \mathbf{b}(\zeta_m^i),$$

where $\zeta_m \in \mathbb{C}$ is an m^{th} root of unity (c.f. [LPR13b, Sections 2.2, 2.5.2]).⁴

With the above definition of inner product (which is in fact the usual inner product over $\mathbb{C}^{\phi(m)}$ when elements in F are represented via the canonical embedding), the norm of an element $\mathbf{b} \in F$ is

$$\|\mathbf{b}\| = \sqrt{\sum_{1 \leq i \leq m, \gcd(i, m)=1} |\mathbf{b}(\zeta_m^i)|^2}.$$

Since all ζ_m^i , where $\gcd(i, m) = 1$, are roots of $\Phi_m(X)$ and $|\zeta_m^i| = 1$, one can check that for any $\mathbf{b} \in F$, $\|\mathbf{b}\| = \|\mathbf{b}X\|$. Since the function $r : F \rightarrow F$

³ The canonical embedding of a polynomial $\mathbf{b} \in F$ is a vector in $\mathbb{C}^{\phi(m)}$ whose coefficients are the evaluations of \mathbf{b} on each of the $\phi(m)$ complex roots of $\Phi_m(X)$. Due to the fact that half of the roots of $\Phi_m(X)$ are conjugates of the other half, the resulting embedded vector in $\mathbb{C}^{\phi(m)}$ can be represented by $\phi(m)$ real numbers.

⁴ We point out that the actual computation of the inner product does not require any operations over \mathbb{C} . The reason is that $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{1 \leq i \leq m, \gcd(i, m)=1} \overline{\mathbf{a}(\zeta_m^i)} \cdot \mathbf{b}(\zeta_m^i)$ can be rewritten as $\overline{(\mathbf{V}\mathbf{a})^T} \mathbf{V}\mathbf{b} = \mathbf{a}^T \overline{\mathbf{V}^T} \mathbf{V}\mathbf{b}$ for a Vandermonde matrix \mathbf{V} with coefficients in \mathbb{C} . The matrix $\overline{\mathbf{V}^T} \mathbf{V}$, however, is a simple *integer* matrix, multiplication by which can be performed in linear time for most “interesting” cyclotomic polynomials (e.g. m is prime or a power of 2).

defined as $r(\mathbf{b}) = \mathbf{b}X$ is linear, it is also an isometry (since it preserves the norm). Furthermore, since F is a field, for any non-zero $\mathbf{b} \in F$, the elements $\mathbf{b}, \mathbf{b}X, \mathbf{b}X^2, \dots, \mathbf{b}X^{\phi(m)-1}$ are all linearly-independent. When \mathbf{b} is an element of $R = \mathbb{Z}[X]/\langle \Phi_m(X) \rangle$, the set

$$\{\mathbf{b}, \mathbf{b}X, \mathbf{b}X^2, \dots, \mathbf{b}X^{\phi(m)-1}\} = \{\mathbf{b}, r(\mathbf{b}), r^2(\mathbf{b}), \dots, r^{\phi(m)-1}(\mathbf{b})\}$$

therefore generates the ideal $\langle \mathbf{b} \rangle$ as an additive group. Such bases containing short elements can serve as private keys in cryptographic schemes.⁵

Paper Organization. In Section 2, we set up the notations and definitions that will be used throughout the paper. In Section 3.1, we describe a simple version of our new algorithm that efficiently orthogonalizes a given set of vectors, and in Section 3.2 we give the full, optimized algorithm. In Section 4, we describe an algorithm that, given the orthogonalization, returns the transformation matrix μ that converts the set $\{\mathbf{b}, r(\mathbf{b}), \dots, r(\mathbf{b})\}$ to $\{\tilde{\mathbf{b}}_1, \tilde{\mathbf{b}}_2, \dots, \tilde{\mathbf{b}}_n\}$. In Section 5, we extend our basic algorithms to those that can more efficiently orthogonalize sets of vectors of the form $\mathbf{b}_1, r(\mathbf{b}_1), \dots, r^{n-1}(\mathbf{b}_1), \mathbf{b}_2, r(\mathbf{b}_2), \dots, r^{n-1}(\mathbf{b}_2), \mathbf{b}_3, r(\mathbf{b}_3), \dots$. These types of sets are the ones that normally occur as secret keys in lattice cryptography. A particular example of such a set is the NTRU lattice, which we discuss in Section 7. In that section, we also give timing comparisons between the *exact* version of our orthogonalization algorithm (which is analyzed in Section 6), and that of [GHN06], for computing the Gram-Schmidt orthogonalization of NTRU lattices. Since the two algorithms use different techniques to achieve speed-ups, we demonstrate that the two improvements can complement each other in the form of an even faster algorithm. In Section 8, we show how to implement Babai's nearest plane algorithm and the Klein/GPV sampling in linear space for lattices whose basis contains vectors that are related via an isometry. In Section 9 we focus on the implementation aspects of our results. In particular, we analyze the required precision to insure the correct functionality of our sampling algorithm.

1.2 Related Work

Computing faster orthogonalization for vectors that are somehow related has been considered in the past. For example, Sweet [Swe84] demonstrated an algorithm that orthogonalizes $d \times d$ Toeplitz matrices using $O(d^2)$ operations. This is the same linear-time speed-up as for our algorithm, but for a different class of

⁵ Normally, the bases used in schemes have slightly different forms, such as consisting of a concatenation of elements from R , or being formed by several elements in R . Such bases still contain large components that are related via an isometry, and we discuss this in more detail in Section 7.

structured matrices.⁶ The techniques in that paper seem to be rather different than in ours – [Swe84] works with the concrete representation of Toeplitz matrices, whereas we only rely on the abstract geometric properties of isometries.

For the special case of NTRU lattices, Gama, Howgrave-Graham, and Nguyen [GHN06] devised algorithms that take advantage of a structure of NTRU bases called *symplecticity*. This allows them to be faster (by a constant factor) than standard Gram-Schmidt orthogonalization when performing orthogonalization in exact arithmetic. We adapt our algorithms for the same application and they outperform those from [GHN06].⁷ And since our algorithm and that of [GHN06] relies on different ideas, it turns out that we can combine the two techniques to achieve a greater overall improvement (see Figure 1 in Section 7).

2 Preliminaries

2.1 Notations

Throughout the paper, we will be working over a d -dimensional inner product space H (usually $H = \mathbb{R}^d$ or \mathbb{C}^d), with $\langle \cdot, \cdot \rangle$ and $\|\cdot\|$ being a scalar product and the associated norm over H . Except when stated otherwise, vectors will be written in bold, matrices and bases in capital bold, and scalars in non-bold letters. $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ will be either an ordered set of independent vectors, also called a basis, or the $n \times d$ matrix whose rows are the \mathbf{b}_i . We denote $\mathbf{B}_k = \text{Span}(\mathbf{b}_1, \dots, \mathbf{b}_k)$ to be the vector space spanned by the vectors $\mathbf{b}_1, \dots, \mathbf{b}_k$.

Definition 1. A linear isometry is a linear map $r : H \rightarrow H$ such that for any $\mathbf{x}, \mathbf{y} \in H$:

$$\langle r(\mathbf{x}), r(\mathbf{y}) \rangle = \langle \mathbf{x}, \mathbf{y} \rangle,$$

or equivalently

$$\|\mathbf{x}\| = \|r(\mathbf{x})\|.$$

For conciseness, we will sometimes say isometry instead of linear isometry. Since the dimension of H is finite, it is immediate that r is invertible. We will be assuming throughout the work that both r and r^{-1} are computable in time $\mathcal{O}(d)$.

Definition 2. Let $\mathbf{x} \in H$, and F be a subspace of H . Then $\text{Proj}(\mathbf{x}, F)$, the projection of \mathbf{x} over F , is the unique vector $\mathbf{y} \in F$ such that $\|\mathbf{x} - \mathbf{y}\| = \min_{\mathbf{z} \in F} \|\mathbf{x} - \mathbf{z}\|$

⁶ One may imagine that it may be possible to somehow adapt the results of [Swe84] to the orthogonalization of bases of ideal lattices. The idea would be to embed elements of $F = \mathbb{Q}[X]/\langle \Phi_m(X) \rangle$ into $C = \mathbb{Q}[X]/\langle X^m - 1 \rangle$ and then try to use the fact that in the coefficient representation, the elements $\mathbf{b}, \mathbf{b}X, \mathbf{b}X^2, \dots$ in C form a Toeplitz matrix. One would have to also take care to make sure that the norm in F corresponds to the coefficient norm in C . We are not sure whether this direction is viable, and even if it is, the algorithm would necessarily involve computations over dimension m , rather than $\phi(m)$, which would unnecessarily increase the running-time of all algorithms.

⁷ We mention that [GHN06] also contains other results which are independent of Gram-Schmidt orthogonalization and are therefore not improved by our work.

Proposition 1. Let r be an isometry and F be a subspace of H . Then :

1. $\mathbf{x} \perp F \Rightarrow r(\mathbf{x}) \perp r(F)$
2. $r(\mathbf{Proj}(\mathbf{x}, F)) = \mathbf{Proj}(r(\mathbf{x}), r(F))$

Proof. We prove the two claims separately :

1. Since r preserves the dot product, it also preserves orthogonality between vectors.
2. r preserves the norm, so

$$\|\mathbf{x} - \mathbf{y}\| = \min_{\mathbf{z} \in F} \|\mathbf{x} - \mathbf{z}\| \implies \|r(\mathbf{x}) - r(\mathbf{y})\| = \min_{\mathbf{z} \in r(F)} \|r(\mathbf{x}) - \mathbf{z}\|$$

□

2.2 The Gram-Schmidt Orthogonalization

We provide three equivalent definitions of the Gram-Schmidt orthogonalization of an ordered basis. The first one is geometrical, the second one is a mathematical formula, and the third one looks at each vector of the basis as its decomposition over two orthogonal vector spaces. Although the two first definitions are standard and useful for comprehension and computation, the third one is less common and we will mostly use it to prove that a basis is indeed the orthogonalization of another one.

Definition 3. Let the basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ be an ordered set of vectors in H . Its Gram-Schmidt orthogonalization (GSO) is the unique basis $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$ verifying one of these properties :

- $\forall k \in \llbracket 1, n \rrbracket, \tilde{\mathbf{b}}_k = \mathbf{b}_k - \mathbf{Proj}(\mathbf{b}_k, \mathbf{B}_{k-1})$
- $\forall k \in \llbracket 1, n \rrbracket, \tilde{\mathbf{b}}_k = \mathbf{b}_k - \sum_{j=1}^{k-1} \frac{\langle \mathbf{b}_k, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2} \tilde{\mathbf{b}}_j$
- $\forall k \in \llbracket 1, n \rrbracket, \tilde{\mathbf{b}}_k \perp \mathbf{B}_{k-1}$ and $(\mathbf{b}_k - \tilde{\mathbf{b}}_k) \in \mathbf{B}_{k-1}$

The bases \mathbf{B} and $\tilde{\mathbf{B}}$ then satisfy : $\forall k \in \llbracket 1, n \rrbracket, \mathbf{B}_k = \tilde{\mathbf{B}}_k$. For a vector \mathbf{b}_k , we will say that $\tilde{\mathbf{b}}_k$ is its Gram-Schmidt reduction (GSR).

Algorithm 1 describes the Gram-Schmidt process as it is usually presented.

2.3 The Gram-Schmidt and LQ Decompositions

The Gram-Schmidt decomposition (GSD) is a natural side-product of the Gram-Schmidt orthogonalization which gives the relation between the input and output bases in the form of a matrix μ . Such a matrix is useful in many cases – for example, it is crucially used in the LLL algorithm [LLL82]. Outside cryptography, applications include solving undetermined linear systems, least square problems and computing eigenvalues.

Algorithm 1. GramSchmidt_Process(\mathbf{B})**Require:** Basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ **Ensure:** Gram-Schmidt reduced basis $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$

- 1: **for** $i = 1, \dots, n$ **do**
- 2: $\tilde{\mathbf{b}}_i \leftarrow \mathbf{b}_i$
- 3: **for** $j = 1, \dots, i - 1$ **do**
- 4: $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2}$
- 5: $\tilde{\mathbf{b}}_i \leftarrow \tilde{\mathbf{b}}_i - \mu_{i,j} \tilde{\mathbf{b}}_j$
- 6: **end for**
- 7: **end for**
- 8: **return** $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$

Definition 4 (Gram-Schmidt Decomposition). Let \mathbf{B} be a $d \times n$ matrix. \mathbf{B} can be uniquely decomposed as $\mathbf{B} = \mu \times \tilde{\mathbf{B}}$, where $\tilde{\mathbf{B}}$ is the GSO of \mathbf{B} and $\mu = (\mu_{i,j})_{1 \leq i, j \leq n}$ is the lower triangular matrix such that

$$\mu_{i,j} = \begin{cases} \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2} & \text{if } i > j \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Notice that the matrix μ is automatically constructed in Algorithm 1 while computing the GSO. This, however, will not be the case in our improved GSO algorithm, and this is why in this paper we will differentiate between GSO and GSD.

We now recall the definition of LQ decomposition and give its natural relation to the $\mathbf{B} = \mu \times \tilde{\mathbf{B}}$ decomposition.

Definition 5 (LQ Decomposition). Let \mathbf{B} be a square invertible matrix. \mathbf{B} can be decomposed as $\mathbf{B} = L \times Q$, where L is a lower triangular matrix and Q is an orthonormal matrix. If we request the diagonal coefficients of L to be positive, then this decomposition is unique.

Fact 1 Let \mathbf{B} be a square invertible matrix, and $\mathbf{B} = \mu \times \tilde{\mathbf{B}}$ its GSD. An LQ decomposition of \mathbf{B} can be computed in time $\mathcal{O}(n^2)$ by taking $Q = D^{-1} \times \tilde{\mathbf{B}}$ and $L = \mu \times D$, where $D = \text{Diag}(\|\tilde{\mathbf{b}}_1\|, \dots, \|\tilde{\mathbf{b}}_n\|)$.

2.4 Discrete Gaussians

Discrete Gaussians are n -dimensional Gaussians discretized over some lattice Λ .

Definition 6. The n -dimensional Gaussian function $\rho_{\sigma, c} : \mathbb{R}^n \rightarrow (0, 1]$ of standard deviation σ and center c is defined by :

$$\rho_{\sigma, c}(\mathbf{x}) \triangleq \frac{1}{(\sigma\sqrt{2\pi})^n} \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{2\sigma^2}\right)$$

For any lattice $\Lambda \subset \mathbb{R}^n$, $\rho_{\sigma, \mathbf{c}}(\Lambda) \triangleq \sum_{\mathbf{x} \in \Lambda} \rho_{\sigma, \mathbf{c}}(\mathbf{x})$. Normalizing $\rho_{\sigma, \mathbf{c}}(\mathbf{x})$ by $\rho_{\sigma, \mathbf{c}}(\Lambda)$, we obtain the probability distribution function of the discrete Gaussian distribution $D_{\Lambda, \sigma, \mathbf{c}}$.

Gaussian Sampling was introduced in [Kle00, GPV08] as a technique to sample from discrete Gaussian distributions, and has since found numerous applications in lattice-based cryptography. In [GPV08], it requires a basis \mathbf{B} of the lattice Λ being sampled from, as well as the GSO $\tilde{\mathbf{B}}$ of \mathbf{B} .

3 Gram-Schmidt Orthogonalization over Isometric Bases

In this section we present our improved isometric Gram-Schmidt algorithm. In Section 3.1, we present a first simple version which we believe is very intuitive to understand, and then present a slightly faster, more involved, version of it in Section 3.2.

Definition 7. Let $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ be an ordered basis of a lattice $\Lambda \subseteq H$. We say that \mathbf{B} is isometric if there exists an isometry r such that

$$\forall k \in \llbracket 2, n \rrbracket, \mathbf{b}_k = r(\mathbf{b}_{k-1})$$

3.1 A Quadratic-Time Algorithm

We now describe a simple algorithm that computes the GSO of any isometric basis in time $\Theta(nd)$ (or $\Theta(n^2)$ when $n = d$).

We briefly expose the general idea behind the algorithm before presenting it formally. If $\tilde{\mathbf{b}}_k$ is the GSR of \mathbf{b}_k , then $r(\tilde{\mathbf{b}}_k)$ is almost the GSR of \mathbf{b}_{k+1} : it is orthogonal to $\mathbf{b}_2, \dots, \mathbf{b}_k$, but not to \mathbf{b}_1 . However, reducing $r(\tilde{\mathbf{b}}_k)$ with respect to \mathbf{b}_1 would break its orthogonality to $\mathbf{b}_2, \dots, \mathbf{b}_k$, so what we really need to do is to reduce it with respect to $\mathbf{b}_1 - \mathbf{Proj}(\mathbf{b}_1, \text{Span}(\mathbf{b}_2 \dots \mathbf{b}_k))$. Indeed, this latter vector is orthogonal to $\mathbf{b}_2, \dots, \mathbf{b}_k$, so reducing $r(\tilde{\mathbf{b}}_k)$ with respect to it won't break the orthogonality of $r(\tilde{\mathbf{b}}_k)$ to $\mathbf{b}_2, \dots, \mathbf{b}_k$. Fortunately, $\mathbf{b}_1 - \mathbf{Proj}(\mathbf{b}_1, \text{Span}(\mathbf{b}_2 \dots \mathbf{b}_k))$ can itself be updated quickly. Definition 8 and Algorithm 2 formalize this idea.

Definition 8. Let $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ be an ordered basis and $k \in \llbracket 1, n \rrbracket$. We denote $\mathbf{v}_{\mathbf{B}, k} = \mathbf{b}_1 - \mathbf{Proj}(\mathbf{b}_1, r(\mathbf{B}_{k-1}))$. When \mathbf{B} is obvious from the context, we simply write \mathbf{v}_k .

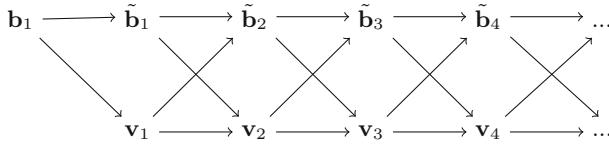
Proposition 2. Let \mathbf{B} be an isometric basis with respect to r . Algorithm 2 returns the GSO of \mathbf{B} . Moreover, if $r(\mathbf{v})$ can be computed in time $O(d)$ for any $\mathbf{v} \in H$, then Algorithm 2 terminates in time $O(nd)$.

Proof. We first prove the correctness of the scheme by proving by induction that for every $k \in \llbracket 1, n \rrbracket$, we have the following :

$$-\mathbf{v}_k = \mathbf{b}_1 - \mathbf{Proj}(\mathbf{b}_1, r(\mathbf{B}_{k-1})) \quad (1)$$

Algorithm 2. Isometric_GSO(\mathbf{B})**Require:** Basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ **Ensure:** Gram-Schmidt reduced basis $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$

- 1: $\tilde{\mathbf{b}}_1 \leftarrow \mathbf{b}_1$
- 2: $\mathbf{v}_1 \leftarrow \mathbf{b}_1$
- 3: **for** $k = 1, \dots, n - 1$ **do**
- 4: $\tilde{\mathbf{b}}_{k+1} \leftarrow r(\tilde{\mathbf{b}}_k) - \frac{\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle}{\|\mathbf{v}_k\|^2} \mathbf{v}_k$
- 5: $\mathbf{v}_{k+1} \leftarrow \mathbf{v}_k - \frac{\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle}{\|\tilde{\mathbf{b}}_k\|^2} r(\tilde{\mathbf{b}}_k)$
- 6: **end for**
- 7: **return** $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$

**Fig. 1.** Computing all the orthogonalized vectors from the first one in Algorithm 2

$$- \tilde{\mathbf{b}}_k = \mathbf{b}_k - \text{Proj}(\mathbf{b}_k, \mathbf{B}_{k-1}) \quad (2)$$

This is trivially true for $k = 1$. Assuming (1) and (2) are true at step k , we have:

- Since \mathbf{v}_k and $\tilde{\mathbf{b}}_k$ are already orthogonal to $r(\mathbf{B}_{k-1})$, \mathbf{v}_{k+1} also is as a linear combination of the two. But \mathbf{v}_{k+1} is also the orthogonalization of \mathbf{v}_k w.r.t. $r(\tilde{\mathbf{b}}_k)$, so it is orthogonal to $r(\mathbf{B}_{k-1}) + \text{Span}(r(\tilde{\mathbf{b}}_k)) = r(\mathbf{B}_k)$. On the other hand, $\mathbf{b}_1 - \mathbf{v}_k$ is in $r(\mathbf{B}_{k-1})$ so $\mathbf{b}_1 - \mathbf{v}_{k+1}$ is in $r(\mathbf{B}_k)$. By applying Definition 3, we can conclude that (1) is true for $k + 1$.
- The same reasoning holds for $\tilde{\mathbf{b}}_{k+1}$: it is orthogonal to $r(\mathbf{B}_{k-1})$ because both \mathbf{v}_k and $r(\tilde{\mathbf{b}}_k)$ are. But since it also is orthogonalized w.r.t. \mathbf{v}_k (in line 4 of the algorithm), it then is orthogonal to $r(\mathbf{B}_{k-1}) + \text{Span}(\mathbf{v}_k) = \mathbf{B}_k$. On the other hand, $\mathbf{b}_{k+1} - \tilde{\mathbf{b}}_{k+1} = r(\mathbf{b}_k - \tilde{\mathbf{b}}_k) + \frac{\langle r(\tilde{\mathbf{b}}_k), \mathbf{v}_k \rangle}{\langle \mathbf{v}_k, \mathbf{v}_k \rangle} \mathbf{v}_k$ is in \mathbf{B}_k . As before, we can conclude that (2) is true for $k + 1$.

Since (2) is verified for any $k \in \llbracket 1, n \rrbracket$, $\tilde{\mathbf{B}}$ is the GSO of \mathbf{B} .

The time complexity of the algorithm is straightforward: assuming additions, subtractions, multiplications and divisions are done in constant time, each scalar product or square norm takes time $\mathcal{O}(d)$. Since there are $3(n-1)$ norms or scalar products, and $2(n-1)$ computations of $r(\cdot)$, the total complexity is $\mathcal{O}(nd)$. \square

3.2 Making Isometric GSO Faster

Algorithm 2 is already $O(n)$ times faster than the classical Gram-Schmidt process. In this subsection, we show that intermediate values are strongly interdependent and that this fact can be used to speed up our GSO implementation by about 67%.

Lemma 1. Let \mathbf{B} be an isometric basis. For any k in $\llbracket 1, n \rrbracket$, we have the following equalities:

- $\langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_k) \rangle = \langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle$
- $\|\mathbf{v}_k\|^2 = \langle \mathbf{v}_k, \mathbf{v}_1 \rangle = \|\tilde{\mathbf{b}}_k\|^2$

When implicit from context, we will denote $C_k = \langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle$ and $D_k = \|\tilde{\mathbf{b}}_k\|^2$. We have the following recursive formula :

$$\forall k \in \llbracket 1, n - 1 \rrbracket, D_{k+1} = D_k - \frac{C_k^2}{D_k}$$

Proof. We prove each of the three equalities separately :

- The equality $\langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_k) \rangle = \langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle$ is equivalent to $\langle \mathbf{v}_k - \mathbf{v}_1, r(\tilde{\mathbf{b}}_k) \rangle = 0$, which is true since $\mathbf{v}_k - \mathbf{v}_1 = \mathbf{Proj}(\mathbf{b}_1, r(\mathbf{B}_{k-1}))$ is in the subspace $r(\mathbf{B}_{k-1})$ and $\tilde{\mathbf{b}}_k$ is orthogonal to $r(\mathbf{B}_{k-1})$
- The equality $\|\mathbf{v}_k\|^2 = \langle \mathbf{v}_k, \mathbf{v}_1 \rangle$ is obtained by following the same reasoning as above
- The equality $\|\mathbf{v}_k\|^2 = \|\tilde{\mathbf{b}}_k\|^2$ is shown by induction : it is the case for $k = 1$. By observing that \mathbf{b}_{k+1} is orthogonal to \mathbf{v}_k from line 4 of Algorithm 2 (resp. \mathbf{v}_{k+1} is orthogonal to $r(\mathbf{b}_k)$ from line 5), we can use the Pythagorean theorem to compute $\|\tilde{\mathbf{b}}_{k+1}\|^2$ and $\|\mathbf{v}_{k+1}\|^2$:

$$\|\tilde{\mathbf{b}}_{k+1}\|^2 = \|\tilde{\mathbf{b}}_k\|^2 - \frac{\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle^2}{\|\mathbf{v}_k\|^2} \quad \text{and} \quad \|\mathbf{v}_{k+1}\|^2 = \|\mathbf{v}_k\|^2 - \frac{\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle^2}{\|\tilde{\mathbf{b}}_k\|^2}$$

At which point we can conclude by induction that $\|\mathbf{v}_{k+1}\|^2 = \|\tilde{\mathbf{b}}_{k+1}\|^2$, and these equalities also yield the recursive formula $D_{k+1} = D_k - \frac{C_k^2}{D_k}$. \square

This result allows us to speed up further the GSO for isometric bases. At each iteration of the algorithm **Isometric_GSO**, instead of computing $\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle$, $\|\tilde{\mathbf{b}}_k\|^2$ and $\|\mathbf{v}_k\|^2$, one only needs to compute $\langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_k) \rangle$, and can instantly compute $\|\tilde{\mathbf{b}}_k\|^2 = \|\mathbf{v}_k\|^2$ from previously known values. We choose $\langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_k) \rangle$ rather than $\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle$ because \mathbf{v}_1 has a much smaller bitsize than \mathbf{v}_k , resulting in a better complexity in exact arithmetic. Moreover, in the case where we use floating-point arithmetic, \mathbf{v}_1 does not introduce any floating-point error, unlike \mathbf{v}_k .

Algorithm 3 sums up these enhancements.

Proposition 3. If \mathbf{B} is an isometric basis, then Algorithm 3 returns the GSO of \mathbf{B} . Moreover, if we disregard the computational cost of r , then Algorithm 3 performs essentially $3n^2$ multiplications (resp. additions), whereas Algorithm 2 performs essentially $5n^2$ multiplications (resp. additions).

Proof. For the correctness of Algorithm 3, one only needs to show that at each step, $C_k = \langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k) \rangle$ and $D_k = \|\tilde{\mathbf{b}}_k\|^2 = \|\mathbf{v}_k\|^2$. The first and third equalities are given by lemma 1, and the second one by induction : assuming that C_k, D_k are correct, D_{k+1} is correct, once again from lemma 1. \square

Algorithm 3. Faster_Isometric_GSO(\mathbf{B})

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$
Ensure: Gram-Schmidt reduced basis $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$, $(C_k)_{1 \leq k < n}$, $(D_k)_{1 \leq k < n}$

- 1: $\tilde{\mathbf{b}}_1 \leftarrow \mathbf{b}_1$
- 2: $\mathbf{v}_1 \leftarrow \mathbf{b}_1$
- 3: $C_1 \leftarrow \langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_1) \rangle$
- 4: $D_1 \leftarrow \|\mathbf{b}_1\|^2$
- 5: **for** $k = 1, \dots, n - 1$ **do**
- 6: $\tilde{\mathbf{b}}_{k+1} \leftarrow r(\tilde{\mathbf{b}}_k) - \frac{C_k}{D_k} \mathbf{v}_k$
- 7: $\mathbf{v}_{k+1} \leftarrow \mathbf{v}_k - \frac{C_k}{D_k} r(\tilde{\mathbf{b}}_k)$
- 8: $C_{k+1} \leftarrow \langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_{k+1}) \rangle$
- 9: $D_{k+1} \leftarrow D_k - \frac{C_k^2}{D_k}$
- 10: **end for**

4 Gram-Schmidt Decomposition over Isometric Bases

In this section, we show that the computation of the matrix μ from the Gram-Schmidt decomposition (or GSD, see Definition 4) can be sped up by a $O(n)$ factor in the case of isometric matrices by using tricks similar to those which led to the speeding-up of GSO. The proof of the following theorem explains how to compute the GSD of an isometric basis/matrix in quadratic time.

Theorem 2. *Let $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ be an isometric basis and $\tilde{\mathbf{B}} = (\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n)$ its GSO. For the sake of simplicity, we identify the basis \mathbf{B} (resp. $\tilde{\mathbf{B}}$) to the (not necessarily square) matrix which rows are the vectors of the basis. Assume we already have \mathbf{B} and $\tilde{\mathbf{B}}$, along with the values $C_j = \langle \mathbf{v}_j, r(\tilde{\mathbf{b}}_j) \rangle$, $D_j = \|\tilde{\mathbf{b}}_j\|^2$ for $1 \leq j < n$. Then the matrix μ associated to \mathbf{B} can be computed in time $O(n^2)$.*

Proof. For $1 \leq i < j \leq n$, let $X_{i,j} = \langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle$ and $Y_{i,j} = \langle r(\mathbf{b}_i), \mathbf{v}_j \rangle$. All the nontrivial values of $\mu_{i,j}$ (that is, the values $\mu_{i,j}$ for $1 \leq j < i \leq n$) can be expressed as $\mu_{i,j} = \frac{X_{i,j}}{D_j}$. The values $X_{i,j}, Y_{i,j}$ satisfy these recursive formulae:

$$\begin{cases} X_{i+1,j+1} = X_{i,j} - \frac{C_j}{D_j} Y_{i,j} \\ Y_{i,j+1} = Y_{i,j} - \frac{C_j}{D_j} X_{i,j} \end{cases}$$

These formulae allow us to compute all the values of $X_{i,j}, Y_{i,j}$ from the $2(n-1)$ values $X_{i,1}, Y_{i,1}$. Once all of these values are computed, one can simply obtain the $\mu_{i,j}$ from the $X_{i,j}$. Algorithm 4 puts this idea into practice.

The idea of this algorithm is somewhat similar to the one behind Algorithms 2 and 3: the only values that we really need to compute are the $X_{i,j}$'s, but in order to do that efficiently we resort to a mutual recursion involving the $Y_{i,j}$'s.

The time complexity is straightforward. Each $X_{i,j}, Y_{i,j}$ takes time $O(1)$ to be computed, except for $2n$ of them which need time $O(n)$ each. So the overall cost is $O(n^2)$. \square

Algorithm 4. Isometric_GSD(\mathbf{B} , $\tilde{\mathbf{B}}$, (C_i) , (D_i))

Require: Basis \mathbf{B} and its orthogonalization $\tilde{\mathbf{B}}$, values $C_j = \langle \mathbf{v}_j, r(\tilde{\mathbf{b}}_j) \rangle$, $D_j = \|\tilde{\mathbf{b}}_j\|^2$ for $1 \leq j < n$

Ensure: Matrix $\mu = \mathbf{B} \times \tilde{\mathbf{B}}^{-1}$

- 1: Set the diagonal values of μ to 1 and the values above the diagonal to 0
- 2: **for** $i = 2 \dots n$ **do** {Computing the $(X_{i,1})$, $(Y_{i,1})$ }
- 3: $X_{i,1} \leftarrow \langle \mathbf{b}_i, \tilde{\mathbf{b}}_1 \rangle$
- 4: $Y_{i,1} \leftarrow \langle r(\mathbf{b}_i), \mathbf{b}_1 \rangle$
- 5: **for** $j = 2 \dots i - 1$ **do**
- 6: $X_{i,j} \leftarrow X_{i-1,j-1} - \frac{C_{j-1}}{D_{j-1}} Y_{i-1,j-1}$
- 7: $Y_{i,j} \leftarrow Y_{i,j-1} - \frac{C_{j-1}}{D_{j-1}} X_{i,j-1}$
- 8: **end for**
- 9: **end for**
- 10: **for** $i = 2 \dots n$ **do** {Filling out the non-trivial values of $\mu_{i,j}$ }
- 11: **for** $j = 1 \dots i - 1$ **do**
- 12: $\mu_{i,j} \leftarrow \frac{X_{i,j}}{D_j}$
- 13: **end for**
- 14: **end for**

As an example, the matrices X_{steps} and X_{chrono} below show, for $n = 5$, in which order the matrices X, Y are filled. The two matrices use different metrics: X_{chrono} displays the chronological order in which the matrices are filled by the algorithm, whereas X_{steps} display the minimal depth of the computational tree necessary in order to compute an $X_{i,j}$ (resp. $Y_{i,j}$). If a box contains \times , it means that the corresponding value is trivial (see step 1 of the algorithm).

$$X_{steps} = \begin{array}{|c|c|c|c|c|} \hline & \times & \times & \times & \times & \times \\ \hline 1 & \times & \times & \times & \times & \times \\ \hline 1 & 2 & \times & \times & \times & \times \\ \hline 1 & 2 & 3 & \times & \times & \times \\ \hline 1 & 2 & 3 & 4 & \times & \times \\ \hline \end{array} \quad X_{chrono} = \begin{array}{|c|c|c|c|c|} \hline & \times & \times & \times & \times & \times \\ \hline 1 & \times & \times & \times & \times & \times \\ \hline 2 & 3 & \times & \times & \times & \times \\ \hline 4 & 5 & 6 & \times & \times & \times \\ \hline 7 & 8 & 9 & 10 & \times & \times \\ \hline \end{array}$$

As X_{chrono} shows, the algorithm fills the matrices X, Y row after row, but if necessary, it could be rewritten in order to fill X, Y column after column, as shown by X_{steps} .

5 Extending the Results to Block Isometric Bases

In previous sections, we showed that we can gain a factor $O(n)$ improvement when performing operations such as Gram-Schmidt decomposition on isometric bases. In this section, we show that these results can be extended to block isometric bases, that is bases that are concatenations of isometric bases.

Definition 9. Let $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_{kn}\}$ be a basis. We say that \mathbf{B} is block isometric if there exist k isometric bases $\mathbf{B}^{(1)}, \dots, \mathbf{B}^{(k)}$ such that \mathbf{B} is the concatenation of all these bases.

Algorithm 5. Block_GSO(\mathbf{B})

Require: Block isometric basis $\mathbf{B} = \{\mathbf{B}^{(1)}, \dots, \mathbf{B}^{(k)}\} = \{\mathbf{b}_1, \dots, \mathbf{b}_{kn}\}$

Ensure: Gram-Schmidt reduced basis $\tilde{\mathbf{B}}$

- 1: **for** $i = 0, \dots, k - 1$ **do**
- 2: $\tilde{\mathbf{b}}_{ni+1} \leftarrow \mathbf{b}_{ni+1}$
- 3: **for** $j = 1, \dots, ni$ **do**
- 4: $\tilde{\mathbf{b}}_{ni+1} \leftarrow \mathbf{b}_{ni+1} - \frac{\langle \mathbf{b}_{ni+1}, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2} \tilde{\mathbf{b}}_j$ {Make $\tilde{\mathbf{b}}_{ni+1}$ orthogonal to previous vectors}
- 5: **end for**
- 6: $\tilde{\mathbf{B}}^{(i+1)} \leftarrow \{\mathbf{b}_{ni+1}, r(\mathbf{b}_{ni+1}), \dots, r^{n-1}(\mathbf{b}_{ni+1})\}$
- 7: $\tilde{\mathbf{B}}^{(i+1)} \leftarrow \text{Faster_Isometric_GSO}(\tilde{\mathbf{B}}^{(i+1)})$
- 8: **end for**

The main idea of Algorithm 5 is to use the hypothesis that $r(Span(\mathbf{B}^{(i)})) = Span(\mathbf{B}^{(i)})$ (which in practice is always verified for ideal lattices) in conjunction with part 2 of Proposition 1 : if $\tilde{\mathbf{b}}$ is the GSR of \mathbf{b} w.r.t. a block $\mathbf{B}^{(i)}$, then $r(\tilde{\mathbf{b}})$ will be the GSR of $r(\mathbf{b})$ w.r.t. that same block $\mathbf{B}^{(i)}$.

Lemma 2. *Assume :*

- $\mathbf{B}^{(1)}, \dots, \mathbf{B}^{(k)}$ are matrices isometric for the same isometry r , and of same rank n
- $\forall i \in \llbracket 1, k - 1 \rrbracket, r(Span(\mathbf{B}^{(i)})) = Span(\mathbf{B}^{(i)})$

Then Algorithm 5 compute the GSO of $\mathbf{B} = \{\mathbf{B}^{(1)}, \dots, \mathbf{B}^{(k)}\} = \{\mathbf{b}_1, \dots, \mathbf{b}_{kn}\}$ in $O(k^2nd)$ elementary operations over the scalars.

Proof. We prove correctness by showing inductively that at the end of each iteration i of the outer loop, the $n(i + 1)$ first vectors $\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_{n(i+1)}$ are the GSO of $\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_{n(i+1)}$:

- For $i = 0$, this is the case since $\tilde{\mathbf{B}}^{(1)}$ is simply the GSO of $\mathbf{B}^{(i)}$
- If it is verified until step $i - 1$, then at step i the vector $\tilde{\mathbf{b}}_{ni+1}$ computed in lines 2-5 of the algorithm is exactly the GSR of \mathbf{b}_{ni+1} . Its rotations are orthogonal to the vectors of the previous blocks because r preserves the dot product and $\forall i, r(Span(\mathbf{B}^{(i)})) = Span(\mathbf{B}^{(i)})$, and one can verify that $\mathbf{b}_{ni+j} - r^{j-1}(\tilde{\mathbf{b}}_{ni+1}) \in Span\{\tilde{\mathbf{B}}^{(1)}, \dots, \tilde{\mathbf{B}}^{(i-1)}\}$, so $r^{j-1}(\tilde{\mathbf{b}}_{ni+1})$ is exactly the orthogonalization of \mathbf{b}_{ni+j} w.r.t. $\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_{ni}$. The basis computed at line 6 is isometric, so applying Faster_Isometric_GSO effectively orthogonalize it.

We now study the complexity of algorithm 5. At each iteration i of the algorithm, the orthogonalization of \mathbf{b}_{ni+1} w.r.t. previous vectors (steps 3 to 5) take time $O(nid)$, and steps 6-7 take time $O(nd)$. So the total complexity is $O(k^2nd)$, gaining a factor n when compared to the complexity $O((kn)^2d)$ of the naive Gram-Schmidt orthogonalisation. \square

The GSD can be sped up too. We will not detail it, but Fig. 2 gives the outline on how to use Algorithm 4 on a two-blocks isometric basis.

$$\tilde{\mathbf{B}} = \begin{bmatrix} \mathbf{B}^{(1)} \\ \mathbf{B}^{(2)} \end{bmatrix} \Rightarrow \begin{bmatrix} \tilde{\mathbf{B}}^{(1)} \\ \mathbf{B}^{(2)} \end{bmatrix} \Rightarrow \begin{bmatrix} \tilde{\mathbf{B}}^{(1)} \\ \{\tilde{\mathbf{b}}_{n+1}, \dots, r^{n-1}(\tilde{\mathbf{b}}_{n+1})\} \end{bmatrix} \Rightarrow \begin{bmatrix} \tilde{\mathbf{B}}^{(1)} \\ \tilde{\mathbf{B}}^{(2)} \end{bmatrix}$$

$$\mu = \begin{bmatrix} I_n | 0_n \\ 0_n | I_n \end{bmatrix} \Rightarrow \begin{bmatrix} \mu_1 | 0_n \\ 0_n | I_n \end{bmatrix} \Rightarrow \begin{bmatrix} \mu_1 | 0_n \\ \mu_3 | I_n \end{bmatrix} \Rightarrow \begin{bmatrix} \mu_1 | 0_n \\ \mu_3 | \mu_4 \end{bmatrix}$$

Fig. 2. Computing the GSD of a two-block isometric basis. $\tilde{\mathbf{B}}$ and μ always satisfy $\mu \times \tilde{\mathbf{B}} = \mathbf{B}$

6 GSO and GSD in Exact Arithmetic

Generally, GSO and GSD are performed over real bases, so the standard way of implementing it is by using floating-point arithmetic. However, this can result in rounding errors: several books and articles discuss this problem with a good introduction being [Hig02].

When the input vectors are in \mathbb{Z}^d , as it is very often the case in lattice-based cryptography, then the GSD can be performed using only exact arithmetic over \mathbb{Q} . Moreover, some algorithms such as the original LLL algorithm [LLL82] explicitly perform exact GSD.

However, this gain in precision comes at the cost of reduced efficiency: when an integer basis undergoes GSO, the reduced vectors' bitsize quickly escalates in the dimension of the basis and of the underlying space. This phenomenon is called *coefficient explosion* and impacts the space *and* computational cost of GSD. In this section, we adapt Algorithms 3 and 4 to the exact arithmetic setting and show that we still gain a $O(d)$ factor compared to classical GSO/GSD. Moreover, our adapted algorithms completely avoid rational arithmetic.

Through this section, we make an additional “niceness” assumption over the isometry r , namely we suppose that it maps integer vectors into integer vectors: $\forall \mathbf{b} \in \mathbb{Z}^d, r(\mathbf{b}) \in \mathbb{Z}^d$.

6.1 GSO in Exact Arithmetic

Definition 10. Let $\mathbf{B} = (\mathbf{b}_j)_{1 \leq j \leq n}$ be an isometric basis, and for $j \in \llbracket 1, n \rrbracket$, $\tilde{\mathbf{b}}_j, \mathbf{v}_j, C_j, D_j$ be defined as in Section 3. We then define, $\forall i, j \in \llbracket 1, n \rrbracket$, the following values:

- $\lambda_{j,j} = \prod_{1 \leq k \leq j} \|\tilde{\mathbf{b}}_k\|^2$
- $\tilde{\mathbf{d}}\mathbf{b}_j = \lambda_{j-1,j-1}\tilde{\mathbf{b}}_j$
- $c_j = \lambda_{j-1,j-1}C_j$
- $\lambda_{i,j} = \mu_{i,j}\lambda_{j,j}$
- $\mathbf{d}\mathbf{v}_j = \lambda_{j-1,j-1}\mathbf{v}_j$
- $d_j = \lambda_{j-1,j-1}D_j$

Proposition 4. Using notations of Definition 10, $\forall i, j \in \llbracket 1, n \rrbracket$, we have:

1. $\lambda_{i,j} \in \mathbb{Z}$
2. $\tilde{\mathbf{d}}\mathbf{b}_j, \mathbf{d}\mathbf{v}_j \in \mathbb{Z}^d$
3. $c_j, d_j \in \mathbb{Z}$

Proof. Proofs for assertions 1 and 2 can be found per example in [Gal12, chapter 17, theorem 17.3.2]. As for assertion 3, $d_j = \lambda_{j,j}$ and $c_j = \langle \mathbf{v}_1, r(\tilde{\mathbf{d}}\mathbf{b}_j) \rangle$, where \mathbf{v}_1 and $r(\tilde{\mathbf{d}}\mathbf{b}_j)$ are in \mathbb{Z}^d . \square

With these results in hand, we can now devise an integer version of Algorithm 3. Instead of outputting rational values, Algorithm 6 outputs only integers and integer vectors, and one can then retrieve any vector $\tilde{\mathbf{b}}_k$ by computing $\mathbf{b}_k = \frac{1}{\sqrt{d_{k-1}}} \tilde{\mathbf{d}}\mathbf{b}_k$. Algorithm 6 uses no rational number and all the internal operations, including exact divisions in steps 6,7 and 9, output integer values. The following lemma shows that in the case we use exact arithmetic, Algorithm 6 is still at least $O(n)$ faster than standard GSO.

Lemma 3. *Let $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\} \in (\mathbb{Z}^d)^n$ be an integral isometric basis, $|\mathbf{B}| = \max_{k=1 \dots n} (\|\mathbf{b}_k\|)$ and $\mathcal{M}(X)$ denote the time complexity for multiplying two integers of at most X bits. Suppose the isometry r associated to \mathbf{B} can be computed in time and space linear to the size of the input. Then Algorithm 6 performs in time $O(dn\mathcal{M}(n \log |\mathbf{B}|))$.*

Proof. By definition, $d_k = \prod_{1 \leq i \leq k} \|\tilde{\mathbf{b}}_i\|^2$ so $|d_k| \leq |\mathbf{B}|^{2k}$. Moreover, $|C_k| < D_k$ implies $|c_k| < d_k$ and therefore c_k, d_k both have bitsizes $O(k \log |\mathbf{B}|)$. On the other hand, $\tilde{\mathbf{d}}\mathbf{b}_k$ (resp. $\mathbf{d}\mathbf{v}_k$) has its norm less than $|\mathbf{B}|^{2k-1}$ so the four scalar-vectors products performed on steps 6,7 have complexity $O(d\mathcal{M}(k \log |\mathbf{B}|))$, as well as the two divisions of vectors by scalars (we recall that euclidean division of X bit numbers can be performed in time $O(\mathcal{M}(X))$). Overall, each iteration k of the **for** loop takes time $O(d\mathcal{M}(k \log |\mathbf{B}|))$, so the total complexity of Algorithm 6 is $O(dn\mathcal{M}(n \log |\mathbf{B}|))$. \square

Algorithm 6. Integer_Isometric_GSO(\mathbf{B})

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$

Ensure: $(\tilde{\mathbf{d}}\mathbf{b}_k, \mathbf{d}\mathbf{v}_k, c_k, d_k)_{k=1 \dots n}$ as defined in Definition 10

- 1: $\tilde{\mathbf{d}}\mathbf{b}_1 \leftarrow \mathbf{b}_1$
 - 2: $\mathbf{d}\mathbf{v}_1 \leftarrow \mathbf{b}_1$
 - 3: $c_1 \leftarrow \langle r(\tilde{\mathbf{b}}_1), \mathbf{d}\mathbf{v}_1 \rangle$
 - 4: $d_1 \leftarrow \|\mathbf{b}_1\|^2$
 - 5: **for** $k = 1, \dots, n - 1$ **do**
 - 6: $\tilde{\mathbf{d}}\mathbf{b}_{k+1} \leftarrow \left[d_k r(\tilde{\mathbf{d}}\mathbf{b}_k) - c_k \mathbf{d}\mathbf{v}_k \right] / d_{k-1}$
 - 7: $\mathbf{d}\mathbf{v}_{k+1} \leftarrow \left[d_k \mathbf{d}\mathbf{v}_k - c_k r(\tilde{\mathbf{d}}\mathbf{b}_k) \right] / d_{k-1}$
 - 8: $c_{k+1} \leftarrow \langle \mathbf{v}_1, r(\tilde{\mathbf{d}}\mathbf{b}_{k+1}) \rangle$
 - 9: $d_{k+1} \leftarrow \frac{d_k^2 - c_k^2}{d_{k-1}}$
 - 10: **end for**
-

6.2 GSD in Exact Arithmetic

The isometric GSD can also naturally be converted into an efficient, “rational-free” version. Let $x_{i,j} \triangleq \lambda_{j-1} X_{i,j} = \langle \mathbf{b}_i, \tilde{\mathbf{d}}\mathbf{b}_j \rangle \in \mathbb{Z}$ and $y_{i,j} \triangleq \lambda_{j-1} Y_{i,j} = \langle r(\mathbf{b}_i), \mathbf{d}\mathbf{v}_j \rangle \in \mathbb{Z}$. The relations

$$\begin{cases} X_{i+1,j+1} = X_{i,j} - \frac{c_j}{D_j} Y_{i,j} \\ Y_{i,j+1} = Y_{i,j} - \frac{c_j}{D_j} X_{i,j} \end{cases}$$

then become

$$\begin{cases} x_{i+1,j+1} = \frac{d_j x_{i,j} - c_j y_{i,j}}{d_{j-1}} \\ y_{i,j+1} = \frac{d_j y_{i,j} - c_j x_{i,j}}{d_{j-1}} \end{cases}$$

The $x_{i,j}$ ’s actually are the $\lambda_{i,j}$ ’s, but in Algorithm 7 we continue to write $x_{i,j}$ since it highlights the natural transformation of Algorithm 4 to Algorithm 7.

Lemma 4. *Following the notations of Lemma 3, the time complexity of Algorithm 7 is $O(n^2 \mathcal{M}(n \log |\mathbf{B}|) + nd\mathcal{M}(\log |\mathbf{B}|))$.*

Proof. The costliest operations of Algorithm 7 are either the $2(n-1)$ dot products in steps 2 and 3, which cost $O(d\mathcal{M}(\log |\mathbf{B}|))$ each, or the essentially $3n^2$ multiplications and divisions made at steps 5 and 6, which cost $O(\mathcal{M}(j \log |\mathbf{B}|))$ each. Summing these costs yields the result. \square

Note that in practice d is not much bigger than n , so the complexity of Algorithm 7 becomes $O(n^2 \mathcal{M}(n \log |\mathbf{B}|))$. Even in exact arithmetic, our GSO and GSD algorithms still perform $O(n)$ times faster than standard GSD, which complexity is $O(dn^2 \mathcal{M}(n \log |\mathbf{B}|))$ (implicit in the proof of [Gal12, Theorem 17.3.4]). Moreover, we manage to avoid the use of any rational number, making our algorithms both efficient and easy to implement.

7 NTRU Lattices

NTRU lattices are a special class of lattices widely used in cryptography, because their *ideal* structure allows a gain of a factor n both in time and space when

Algorithm 7. Integer_Isometric_GSD($\mathbf{B}, (c_k, d_k)_{k=1\dots n}$)

Require: Basis \mathbf{B} and the values $(c_k, d_k)_{k=1\dots n}$

Ensure: $x_{i,j}$ ’s, $y_{i,j}$ ’s as defined above

- 1: **for** $i = 2\dots n$ **do** {Computing the $(x_{i,1}), (y_{i,1})$ }
 - 2: $x_{i,1} \leftarrow \langle \mathbf{b}_i, \mathbf{b}_1 \rangle$
 - 3: $y_{i,1} \leftarrow \langle r(\mathbf{b}_i), \mathbf{b}_1 \rangle$
 - 4: **for** $j = 2\dots i-1$ **do**
 - 5: $x_{i,j} \leftarrow [d_{j-1}x_{i-1,j-1} - c_{j-1}y_{i-1,j-1}] / d_{j-2}$
 - 6: $y_{i,j} \leftarrow [d_{j-1}y_{i,j-1} - c_{j-1}x_{i,j-1}] / d_{j-2}$
 - 7: **end for**
 - 8: **end for**
-

performing usual operations over lattices. This results in efficient and compact cryptosystems (e.g. [HPS98, LTV12, DDLL13]).

Definition 11. Let $N, q \in \mathbb{N}^*$ and $f, g, F, G \in \mathbb{Z}_N[x]$ such that $fG - gF = q \pmod{(x^N + 1)}$. The NTRU lattice generated by f, g, F, G is the lattice generated by the rows of the block matrix

$$\left[\begin{array}{c|c} \mathcal{A}(f) & \mathcal{A}(g) \\ \hline \mathcal{A}(F) & \mathcal{A}(G) \end{array} \right]$$

Where $\mathcal{A}(p)$ is the $N \times N$ matrix which i -th row is the coefficients of $x^{i-1} \cdot p(x) \pmod{(x^N + 1)}$.

In [GHN06], Gama et al. considered exact GSD of NTRU bases. They showed that these lattices verify an algebraic property called symplecticity, which allows them to compute the exact GSD faster than with the standard algorithm, using [GHN06, Corollary 1].

But in addition to being q -symplectic, NTRU bases are also block isometric. So we devised an algorithm to compute the exact GSD of a NTRU basis, by combining three strategies:

- use Algorithms 6 and 7 in order to avoid rational arithmetic (as in [Gal12] and [GHN06])
- use the GSO/GSD strategies for isometric bases detailed in Section 5
- use [GHN06, Corollary 1] to compute only one half of the GSO and get the other for free

We compared our exact reduction algorithm with the ones from [GHN06]. It turns out that our algorithm is faster, both theoretically and in practice, despite computing more information: it provides $\tilde{\mathbf{B}}$ and μ , whereas the algorithms in [GHN06] only provide μ . The timings are summarized in Table 1 and the full implementation can be found at <https://github.com/tprest/Fast-GSD>.

Table 1. Timings for Gram-Schmidt over NTRU bases, in seconds. The implementation was done on **Sage** 5.3. Timings were performed on an Intel Core i5-3210M laptop with a 2.5GHz CPU and 6GB RAM. Isometric GSD is “standard” GSD for block isometric bases, whereas Iso.+Symp. GSD takes into account the observations from [GHN06].

Dimension $n = 2N$	128	256	512	1024
Standard GS [GHN06]	3.22	30.7	390	4536
Dual GS [GHN06]	2.39	17	214	2496
Symplectic GS [GHN06]	0.89	5.73	33.9	279
Isometric GSD	0.48	2.05	12.4	89
Iso.+Symp. GSD	0.312	1.4	8.18	57.8

8 Reversibility and Application to Linear-Storage Gaussian Sampling

Gaussian Sampling [Kle00, GPV08] is a cornerstone of lattice cryptography. It can either serve to find approximately close lattice points close to a vector [Kle00], or to sample a lattice point close to a target point without leaking any information about the basis used [GPV08]. We recall the definition of the Gaussian Sampler:

A drawback of applying the Gaussian Sampler over ideal lattices is that, even though the basis \mathbf{B} of an ideal lattice can be stored using $O(1)$ vectors, this is not the case for the reduced basis $\tilde{\mathbf{B}}$, which needs n vectors. This can quickly impede the practicality of the Gaussian Sampler: for example, for $n = 1024$ (a typical dimension for cryptographic lattices), if $\tilde{\mathbf{B}}$ is stored using 128 bits of precision, the bitsize of $\tilde{\mathbf{B}}$ then exceeds 128 Mbits.

Our algorithm allows to overcome this problem by computing the reduced basis $\tilde{\mathbf{B}}$ on-the-fly. An obstacle is that Gaussian Sampling needs the vectors of $\tilde{\mathbf{B}}$ in reverse order, so a straightforward use of Algorithm 2 or 3 does not solve the problem since it provides the basis in direct order. Fortunately, as Fig. 1 suggests, Algorithms 2 and 3 can be “reversed” in the sense that provided with the last vector $\tilde{\mathbf{b}}_n$ of the basis $\tilde{\mathbf{B}}$ and a few extra pieces of information, one can compute $\tilde{\mathbf{b}}_{n-1}, \dots, \tilde{\mathbf{b}}_1$ on-the-fly.

Definition 12. For a basis \mathbf{B} , we denote, for any $i \in \llbracket 1; n-1 \rrbracket$, $C_i = \langle \mathbf{v}_i, r(\tilde{\mathbf{b}}_i) \rangle$ and $D_i = \|\tilde{\mathbf{b}}_i\|^2$. We also define $H_i = \frac{1}{1-(C_i/D_i)^2} = \frac{D_i}{D_{i+1}}$ and $I_i = \frac{C_i/D_i}{1-(C_i/D_i)^2} = \frac{C_i}{D_{i+1}}$.

Lemma 5. Algorithms 8 and 9 produce the same output when they have the same input \mathbf{B} and \mathbf{c} (assuming the associated precomputed values are correct).

Proof. First, observe that $\forall i = 1 \dots n-1$, $|C_i| < D_i$, because otherwise D_{i+1} would be zero and \mathbf{B} would not be a basis. One can see that the “linear system”

$$\begin{aligned} - \tilde{\mathbf{b}}_{i+1} &= r(\tilde{\mathbf{b}}_i) - \frac{C_i}{D_i} \mathbf{v}_i \\ - \mathbf{v}_{i+1} &= \mathbf{v}_i - \frac{C_i}{D_i} r(\tilde{\mathbf{b}}_i) \end{aligned}$$

is invertible :

$$\begin{aligned} - \tilde{\mathbf{b}}_i &= r^{-1}(H_i \tilde{\mathbf{b}}_{i+1} + I_i \mathbf{v}_{i+1}) \\ - \mathbf{v}_i &= I_i \tilde{\mathbf{b}}_{i+1} + H_i \mathbf{v}_{i+1} \end{aligned}$$

H_i and I_i are always defined since $|C_i| < D_i$. Therefore, the same way the values C_i, D_i allow to compute $\tilde{\mathbf{b}}_{i+1}, \mathbf{v}_{i+1}$ from $\mathbf{b}_i, \mathbf{v}_i, H_i, I_i$ allow to compute $\tilde{\mathbf{b}}_i, \mathbf{v}_i$ from $\tilde{\mathbf{b}}_{i+1}, \mathbf{v}_{i+1}$. \square

This allows us to perform Gaussian Sampling using $O(m)$ memory space instead of $O(mn)$ for the classic version. The overhead in time is reasonable :

- Classic Sampler: $2mn$ additions, $2mn$ multiplications, n samplings in \mathbb{Z}

- Compact Sampler: $4mn$ additions, $6mn$ multiplications, n samplings in \mathbb{Z}

Therefore, the compact Gaussian Sampler is at most three times slower than the classic one. This is confirmed by experiments summarised in Table ???. Moreover,

Algorithm 8. Gaussian_Sampler($\mathbf{B}, \tilde{\mathbf{B}}, \sigma, \mathbf{c}$)

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$, its GSO $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$, $\sigma > 0$, center $\mathbf{c} \in \mathbb{Z}^m$

Ensure: \mathbf{z} sampled in $\mathcal{D}_{A(\mathbf{B}), \sigma, \mathbf{c}}$

- 1: $\mathbf{c}_n \leftarrow \mathbf{c}$
 - 2: **for** $i \leftarrow n, \dots, 1$ **do**
 - 3: $d_i \leftarrow \langle \mathbf{c}_i, \tilde{\mathbf{b}}_i \rangle / \|\tilde{\mathbf{b}}_i\|^2$
 - 4: $\sigma_i \leftarrow \sigma / \|\tilde{\mathbf{b}}_i\|$
 - 5: $z_i \leftarrow D_{\mathbb{Z}, \sigma_i, d_i}$
 - 6: $\mathbf{c}_{i-1} \leftarrow \mathbf{c}_i - z_i \tilde{\mathbf{b}}_i$
 - 7: **end for**
 - 8: **return** $\mathbf{c} - \mathbf{c}_0$
-

Algorithm 9. Compact_Gaussian_Sampler($\mathbf{B}, \tilde{\mathbf{B}}, \sigma, \mathbf{c}, \tilde{\mathbf{b}}_n, \mathbf{v}_n, \mathbf{H}, \mathbf{I}$)

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$, center $\mathbf{c} \in \mathbb{Z}^m$, precomputed vectors $\tilde{\mathbf{b}}_n, \mathbf{v}_n$, pre-computed values $(H_i, I_i)_{1 \leq i < n}$ from definition 12

Ensure: \mathbf{z} sampled in $\mathcal{D}_{A(\mathbf{B}), \sigma, \mathbf{c}}$

- 1: $\mathbf{c}_n \leftarrow \mathbf{c}$
 - 2: **for** $i \leftarrow n, \dots, 1$ **do**
 - 3: $d_i \leftarrow \langle \mathbf{c}_i, \tilde{\mathbf{b}}_i \rangle / \|\tilde{\mathbf{b}}_i\|^2$
 - 4: $\sigma_i \leftarrow \sigma / \|\tilde{\mathbf{b}}_i\|$
 - 5: $z_i \leftarrow D_{\mathbb{Z}, \sigma_i, d_i}$
 - 6: $\mathbf{c}_{i-1} \leftarrow \mathbf{c}_i - z_i \tilde{\mathbf{b}}_i$
 - 7: $\tilde{\mathbf{b}}_{i-1} \leftarrow r^{-1}(H_{i-1} \tilde{\mathbf{b}}_i + I_{i-1} \mathbf{v}_i)$
 - 8: $\mathbf{v}_{i-1} \leftarrow I_{i-1} \tilde{\mathbf{b}}_i + H_{i-1} \mathbf{v}_i$
 - 9: **end for**
 - 10: **return** $\mathbf{c} - \mathbf{c}_0$
-

in Algorithm 9, it is possible to sample around several \mathbf{c} 's at the same time: this then makes negligible the overhead induced by the addition (in Algorithm 9) of lines 7 and 8. This time-memory trade-off allows to do Gaussian Sampling for k targets in space $O(km)$ and in time *at most* $(1 + \frac{2}{k})$ times the time required by the classic Gaussian Sampler.

8.1 Analysis of the Space Requirement for the Gaussian Sampler

Suppose that $\tilde{\mathbf{B}}$ needs to be known up to $\lceil \log_2 \epsilon \rceil$ bits, for some $\epsilon < 1$. In order to be able to run Algorithm 9 any time (without having to undergo the GSO beforehand), one only needs to store $(H_i, I_i, \|\tilde{\mathbf{b}}_i\|)_{i=1 \dots n}$ as well as $\tilde{\mathbf{b}}_n, \mathbf{v}_n$. However, it is straightforward to use the relation $\frac{\|\tilde{\mathbf{b}}_{i+1}\|^2}{\|\tilde{\mathbf{b}}_i\|^2} = 1 - \left(\frac{C_i}{D_i}\right)^2$ to save even more space by just storing the $\|\tilde{\mathbf{b}}_i\|$'s and deriving the H_i 's, I_i 's from them. During the execution of Algorithm 9, one also needs to store the current $\tilde{\mathbf{b}}_i, \mathbf{v}_i$. So overall the space requirement of Algorithm 9 is $5n(\lceil \log_2 \epsilon \rceil + b)$, where b is less the “number of bits lost” in steps 6, 7 of Algorithm 9: in other words, b is such that if $\tilde{\mathbf{b}}_n, \mathbf{v}_n, (\|\tilde{\mathbf{b}}_i\|)_{i=1 \dots n}$ are known up to $\lceil \log_2 \epsilon \rceil + b$ bits, then $\tilde{\mathbf{B}}$ is guaranteed to be known up to $\lceil \log_2 \epsilon \rceil$ bits.

For NTRU lattices, this analysis can be refined: only half of the $\|\tilde{\mathbf{b}}_i\|$ need to be known, and $\tilde{\mathbf{b}}_n, \mathbf{v}_n$ can be determined from $\mathbf{b}_1 = \tilde{\mathbf{b}}_1$ [GHN06, Corollary 1]. Instead of needing to know $3n(|\log_2 \epsilon| + b)$ bits beforehand, we just need $\frac{n}{2}(|\log_2 \epsilon| + b)$, so the total space requirement is $2.5n(|\log_2 \epsilon| + b)$.

9 Precision of the Gaussian Sampler

It is known [GPV08] that for σ big enough, the output f of Algorithm 8 is statistically close to the distribution $\mathcal{D}_{A(\mathbf{B}), \sigma, \mathbf{c}}$. However, the proof holds only when $\mathbf{B}, \tilde{\mathbf{B}}, \sigma, \mathbf{c}$ and the values $\|\tilde{\mathbf{b}}_i\|$'s are known exactly. But in practice, one can not afford to do computations with the exact representation of $\tilde{\mathbf{B}}$ and of the $\|\tilde{\mathbf{b}}_i\|$'s, as it would be too costly in terms of space and computational resources. Therefore, $\tilde{\mathbf{B}}$ and the $\|\tilde{\mathbf{b}}_i\|$'s are stored up to some finite precision, and this finite precision introduces errors ϵ, δ_1 which impact the output distribution of the algorithm. Theorem 3 bounds the statistical distance $\Delta(f, f_{\epsilon, \delta_1})$ between the output distribution f of the “perfect” algorithm, and the output distribution f_{ϵ, δ_1} of the “imperfect” algorithm.

Theorem 3. *Let $m, n, q \in \mathbb{N}^*$, $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_n\} \in \mathbb{Z}^{n \times m}$ be a basis of a lattice $A \subseteq \mathbb{Z}^m$, $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \dots, \tilde{\mathbf{b}}_n\}$ be the exact GSO of \mathbf{B} and $\mathbf{c} \in \mathbb{Z}_q^m$. Let $\delta, \epsilon, k > 0$, $\sigma \geq \max \|\tilde{\mathbf{b}}_i\|$ and for any $i = 1 \dots n$, let $\sigma_i = \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|}$. Let f_{ϵ, δ_1} be the output distribution of Algorithm 8 ran on input $(\mathbf{B}, \tilde{\mathbf{B}}, \sigma, (\|\tilde{\mathbf{b}}_i\|)_i, \mathbf{c})$, where the coefficients of $\tilde{\mathbf{B}}$ are known with absolute precision at least δ_1 , and the values $\|\tilde{\mathbf{b}}_i\|$ are known with relative precision at least ϵ . When $\tilde{\mathbf{B}}$ and the $\|\tilde{\mathbf{b}}_i\|$ are known exactly, we simply refer to the output distribution as f .*

Let $\delta_3 = \frac{2k\epsilon q\sqrt{m}}{\sigma} + \epsilon k^2 + \frac{mq\delta_1 k}{\sigma \min_i \|\tilde{\mathbf{b}}_i\|}$. If $\delta_3 \leq 1/2$, then:

$$\Delta(f, f_{\epsilon, \delta_1}) \leq 2n(\delta_3 + 3e^{-k^2/2})$$

In particular, if we want $\Delta(f, f_{\epsilon, \delta_1})$ to be less than $2^{-\lambda}$ for some $\lambda > 0$, it is enough that:

$$\delta_1, \epsilon \leq \frac{2^{-\lambda}}{2n\sqrt{\lambda + \log_2 n} \left(\sqrt{\lambda + \log_2 n} + \frac{2q\sqrt{m}}{\sigma} + \frac{mq}{\sigma \min_i \|\tilde{\mathbf{b}}_i\|} \right)}$$

Proof. f is the output distribution of Algorithm 8 executed on exact input $(\mathbf{B}, \tilde{\mathbf{B}}, \sigma, (\|\tilde{\mathbf{b}}_i\|)_i, \mathbf{c})$, and f_{ϵ, δ_1} is the output distribution of Algorithm 8 executed on input $(\mathbf{B}, \tilde{\mathbf{B}}', \sigma, (\|\tilde{\mathbf{b}}_i\|')_i, \mathbf{c})$, where:

- each vector $\tilde{\mathbf{b}}'_i$ of $\tilde{\mathbf{B}}'$ is correct up to $\log_2 \delta_1$ bits after the comma:
 $\forall i, \|\tilde{\mathbf{b}}_i - \tilde{\mathbf{b}}'_i\|_\infty \leq \delta_1$
- each $\|\tilde{\mathbf{b}}_i\|'$ is correct up to $\log_2 \epsilon$ bits: $\left| \frac{\|\tilde{\mathbf{b}}_i\|}{\|\tilde{\mathbf{b}}_i\|'} - 1 \right| \leq \epsilon$
- since $\sigma_i = \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|}$, $\sigma'_i = \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|'}$, each σ'_i is also correct up to $\log_2 \epsilon$ bits

Let $g = f_{\epsilon, \delta_1}$. Our goal is to make f and g fall into the conditions of Lemma 6 for some X, X', γ, δ_5 , so that we can get a bound on their statistical distance.

Let $X = \{\sum_{i=1\dots n} z_i \mathbf{b}_i | (z_1, \dots, z_n) \in \mathbb{Z}^n\}$. Each possible output $\mathbf{z} = \sum_i z_i \mathbf{b}_i$ implicitly defines (d_1, \dots, d_n) . Let $X' = \{\mathbf{z} = \sum_i z_i \mathbf{b}_i | \forall i, |z_i - d_i| \leq k\sigma_i\}$. From [Lyu12, Lemma 4.4, part 1], $f(X \setminus X'), g(X \setminus X') \leq 1 - (1 - 4e^{-k^2/2})^n \leq 4ne^{-k^2/2}$. On the other hand, Lemma 10 tells us that for $\delta_4 \approx 4\delta_3 + 4e^{-k^2/2}$, $\forall \mathbf{z} = \sum_{i=1\dots n} z_i \mathbf{b}_i \in X$ and $\forall i = 1\dots n$:

$$1 - \delta_4 \leq \frac{D_{\mathbb{Z}, \sigma'_i, d'_i}(z_i)}{D_{\mathbb{Z}, \sigma_i, d_i}(z_i)} \leq 1 + \delta_4$$

Combining this with $f(\mathbf{z}) = \prod_{i=1\dots n} D_{\mathbb{Z}, \sigma_i, d_i}(z)$ and $g(\mathbf{z}) = \prod_{i=1\dots n} D_{\mathbb{Z}, \sigma'_i, d'_i}(z)$, we have a bounded ratio $\frac{g}{f}$ over X' :

$$\forall \mathbf{z} \in X', 1 - n\delta_4 \leq (1 - \delta_4)^n \leq \frac{g(\mathbf{z})}{f(\mathbf{z})} \leq (1 + \delta_4)^n = 1 + n\delta_4 + O(\delta_4^2)$$

Taking $\gamma = 4ne^{-k^2/2}$ and $\delta_5 = n\delta_4$, we can now apply Lemma 6:

$$2\Delta(f, g) \leq n\delta_4 + 2\gamma$$

□

The proof of Theorem 3 resorts to several lemmas that can be found in Appendix A.

9.1 Application to NTRU Lattices

We use the formula obtained in Theorem 3 in order to derive concrete bounds in the case of NTRU lattices. In this particular case, $m = n, \sigma \min \|\tilde{\mathbf{b}}_i\| > q$ [GHN06, Corollary 1] and $\sigma > \sqrt{q}$ (because $\prod_i \|\tilde{\mathbf{b}}_i\| = q^{n/2}$). We can also reasonably assume that $\log_2 n < \lambda < qn/2$, so

$$\epsilon, \delta_1 < \frac{2^{-\lambda}}{8n\sqrt{\lambda qn}} \quad \Rightarrow \quad \Delta(f, f_{\epsilon, \delta_1}) \leq 2^{-\lambda}$$

As an example, for $n = 1024, q \leq 2^{26}$ and $\lambda \leq 256$, this means that $\Delta(f, f_{\epsilon, \delta_1}) \leq 2^{-128}$ provided that $\tilde{\mathbf{B}}$ (resp. the $\|\tilde{\mathbf{b}}_i\|$'s) is known up to $\lambda + 35 + \log_2(\max_i \|\tilde{\mathbf{b}}_i\|)$ (resp. $\lambda + 35$) bits of precision.

We now use the results from Subsection 8.1 to determine the space requirements for the parameters as above. Algorithm 9 requires $2.5n(\lambda + 35 + b)$ bits of space. In experiments we launched on NTRU lattices, we always get $b \leq 30$. We will therefore take this value for b .

As a test for the practicality of our compact Gaussian Sampler, we implemented both the classic and compact Gaussian Samplers and compared their timings and space requirements. As predicted by our computations, the compact Gaussian Sampler is no more than thrice slower than the classic one, while

Table 2. Timings (in milliseconds) and space requirements (in bits and mega-bits) of the classic and compact Gaussian Samplers (Classic GS and Compact GS). The implementation was done in C++ using GMP. Timings were performed on an Intel Core i5-3210M laptop with a 2.5GHz CPU and 6GB RAM.

Statistical distance from ideal		2^{-80}	2^{-128}	2^{-192}
Precision needed	Classic GS	115 bits	163 bits	223 bits
	Compact GS	145 bits	193 bits	253 bits
Running time	Classic GS	115 ms	170 ms	203 ms
	Compact GS	446 ms	521 ms	523 ms
Space requirements	Classic GS	115 Mb	163 Mb	223 Mb
	Compact GS	0.35 Mb	0.47 Mb	0.63 Mb

having space requirements smaller by between two and three orders of magnitude. Our results are summarized in Table 2 and the complete implementation can be found on <https://github.com/tprest/Compact-Sampler>.

Acknowledgments. The authors wish to thank Phong Q. Nguyen, as well as the anonymous Eurocrypt’15 reviewers, for helpful comments which helped improve the presentation of this work.

A Lemmas Used In the Precision Analysis of the Gaussian Sampler

This section regroups the lemmas used by Theorem 3 in order to bound the statistical distance. We will sometimes resort to approximations such as $\rho_{d,\sigma}(\mathbb{Z}) \approx 1$ in order to simplify computations: indeed, $|\rho_{d',\sigma}(\mathbb{Z}) - 1| < 1.04 \cdot 10^{-8\sigma^2}$ whenever $\sigma > \frac{1}{\sqrt{2}}$ (see e.g. [MS07, Sect. 1.1]), and that will always be the case through this section. In the same way, we always assume ϵ and δ_i ’s to be very small and will therefore discard δ terms whenever possible. Each time such an approximation is done, it is indicated with signs such as $O(\cdot)$ or \approx , and has a negligible impact. More precisely, it never adds “hidden errors” to the result being proven.

The first lemma gives a simple bound on the statistical distance between two distributions f and g which are both in a set X' with probability $1 - \gamma$, and enjoy a relative error bound $1 - \delta_5 \leq \frac{g(\mathbf{z})}{f(\mathbf{z})} \leq 1 + \delta_5$ over this set X' . As one could expect, the statistical distance between f and g becomes linear in $\delta_5 + \gamma$ when $\delta_5, \gamma \rightarrow 0$.

Lemma 6. *Let f, g be two distributions over a set X . Let $X' \subseteq X$, $\delta_5, \gamma > 0$ such that $\sum_{\mathbf{z} \in X \setminus X'} f(\mathbf{z}), \sum_{\mathbf{z} \in X \setminus X'} g(\mathbf{z}) \leq \gamma$, and $\forall \mathbf{z} \in X', 1 - \delta_5 \leq \frac{g(\mathbf{z})}{f(\mathbf{z})} \leq 1 + \delta_5$. Then:*

$$2\Delta(f, g) \leq 2\gamma + \delta_5.$$

Proof. We separate the statistical distance sum into two sums over X' and $X \setminus X'$:

$$\begin{aligned} 2\Delta(f, g) &\leq \sum_{\mathbf{z} \in X \setminus X'} f(\mathbf{z}) + \sum_{\mathbf{z} \in X \setminus X'} g(\mathbf{z}) + \sum_{\mathbf{z} \in X'} |f(\mathbf{z}) - g(\mathbf{z})| \\ &\leq 2\gamma + \delta_5 \sum_{\mathbf{z} \in X'} f(\mathbf{z}) \\ &\leq 2\gamma + \delta_5 \end{aligned}$$

□

The following lemma bounds the error occurring in step 3 of Algorithm 8, when the center d_i is computed. In the floating-point version, the $\tilde{\mathbf{b}}_i$ are known up to absolute precision δ_1 (*i.e.* $\log_2 \delta_1$ bits after the comma) and their norms $\|\tilde{\mathbf{b}}_i\|$ are known up to relative precision ϵ (*i.e.* $\log_2 \epsilon - \log_2 \|\tilde{\mathbf{b}}_i\|$ bits after the comma), where δ_1 and ϵ are not necessarily equal.

Lemma 7. Let $0 < \delta_1, \epsilon \ll 1$, $\mathbf{c} \in \mathbb{Z}_q^m$, $\mathbf{b}, \mathbf{b}' \in \mathbb{R}^m$ such that $\|\mathbf{b} - \mathbf{b}'\|_\infty \leq \delta_1$ and $\left| \frac{\|\mathbf{b}\|}{\|\mathbf{b}'\|} - 1 \right| \leq \epsilon$. Let $d = \frac{\langle \mathbf{c}, \mathbf{b} \rangle}{\|\mathbf{b}\|^2}$, $d' = \frac{\langle \mathbf{c}, \mathbf{b}' \rangle}{\|\mathbf{b}'\|^2}$. Then $|d - d'| \leq \delta_2$, where $\delta_2 \approx 2\epsilon \frac{q\sqrt{m}}{\|\mathbf{b}\|} + \frac{mq\delta_1}{\|\mathbf{b}\|^2}$.

Proof. $d' = \left(\frac{\langle \mathbf{c}, \mathbf{b} \rangle}{\|\mathbf{b}\|^2} + \frac{\langle \mathbf{c}, \mathbf{b}' - \mathbf{b} \rangle}{\|\mathbf{b}\|^2} \right) \frac{\|\mathbf{b}\|^2}{\|\mathbf{b}'\|^2}$, so

$$\begin{aligned} |d' - d| &\leq ((1 + \epsilon)^2 - 1)d + \frac{\|\mathbf{c}\|_1 \|\mathbf{b}' - \mathbf{b}\|_\infty}{\|\mathbf{b}\|^2} (1 + \epsilon)^2 \\ &\leq 2 \frac{q\sqrt{m}}{\|\mathbf{b}\|} (\epsilon + O(\epsilon^2)) + \frac{mq}{\|\mathbf{b}\|^2} (\delta_1 + O(\delta_1^2)) \end{aligned}$$

□

In the three next lemmas, we study the difference of behaviour between a perfect gaussian over \mathbb{Z} of center d and standard deviation σ , and the same gaussian with a slightly perturbed center d' and standard deviation σ' . For any center d , $D_{\mathbb{Z}, \sigma, d}$ can be exactly simulated from $D_{\mathbb{Z}, \sigma, d-1}$ (and reciprocally), so we can suppose w.l.o.g. that $d \in (-1/2, 1/2]$. The Lemmas 8, 9 progressively build up to establish in Lemma 10 a bound over the ratio of $D_{\mathbb{Z}, \sigma, d}$ and $D_{\mathbb{Z}, \sigma', d'}$, which are the distributions from which Algorithm 8 samples in step 5 ($D_{\mathbb{Z}, \sigma, d}$ in the “perfect” algorithm, $D_{\mathbb{Z}, \sigma', d'}$ in the “imperfect” one).

Lemma 8. Let $\epsilon, \delta_2, k > 0, \sigma, \sigma' > 1$ and $d, d' \in (-1/2, 1/2]$ such that $|d - d'| \leq \delta_2$ and $\left| \frac{\sigma}{\sigma'} - 1 \right| \leq \epsilon$. Let $z \in \mathbb{Z}$ such that $|z - d| \leq k\sigma$. Then

$$e^{-\delta_3} \leq \frac{\rho_{\sigma', d'}(z)}{\rho_{\sigma, d}(z)} \leq e^{\delta_3}$$

where $\delta_3 = \frac{\delta_2 k}{\sigma} + \epsilon(k^2 + 1) + O(\epsilon, \delta_2^2, \epsilon\delta_2)$. In particular, if $\delta_3 \leq 1/2$, then $\left| \frac{\rho_{\sigma', d'}(z)}{\rho_{\sigma, d}(z)} - 1 \right| \leq 2\delta_3$

Proof.

$$\frac{\rho_{\sigma', d'}(z)}{\rho_{\sigma, d}(z)} = \frac{\rho_{\sigma', d'}(z)}{\rho_{\sigma', d}(z)} \times \frac{\rho_{\sigma, d'}(z)}{\rho_{\sigma, d}(z)}$$

One one hand,

$$\frac{\rho_{\sigma,d'}(z)}{\rho_{\sigma,d}(z)} = e^{\frac{(d'-d)(2z-d'+d)}{2\sigma^2}}, \text{ and } \left| \frac{(d'-d)(2z-d'+d)}{2\sigma^2} \right| \leq \frac{k}{\sigma}(\delta_2 + O(\delta_2^2))$$

On the other hand,

$$\frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d'}(z)} = \frac{\sigma}{\sigma'} \times e^{\frac{(z-d')^2}{2\sigma^2} - \frac{(z-d')^2}{2\sigma'^2}}$$

and

$$\left| \frac{(z-d')^2}{2\sigma^2} - \frac{(z-d')^2}{2\sigma'^2} \right| = \frac{(z-d')^2}{2\sigma^2} \left| 1 - \frac{\sigma^2}{\sigma'^2} \right| \leq k^2(\epsilon + O(\epsilon^2))$$

Combining both inequalities and using $e^x = 1 + x + O(x^2)$ yield:

$$\begin{aligned} \frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d}(z)} &\leq (1 + \frac{k}{\sigma}(\delta_2 + O(\delta_2^2))) (1 + \epsilon) (1 + k^2(\epsilon + O(\epsilon^2))) \\ &\leq 1 + \frac{\delta_2 k}{\sigma} + \epsilon(k^2 + 1) + O(\epsilon, \delta_2^2, \epsilon \delta_2) \end{aligned}$$

□

Lemma 9. Let $\sigma, \sigma' > 1$ and $d, d' \in (-1/2, 1/2]$. Let $k > 0$ and $Z = \{z \in \mathbb{Z}, |z - d| \leq k\sigma\}$. Suppose $\exists \delta_3 \in (0, 1/2)$, $\forall z \in Z$, $\left| \frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d}(z)} - 1 \right| \leq 2\delta_3$. Then:

$$\sum_{z \in \mathbb{Z}} |\rho_{\sigma',d'}(z) - \rho_{\sigma,d}(z)| \leq 2\delta_3 + 4e^{-k^2/2}$$

Proof. We separate the sum in two sums over Z and $\mathbb{Z} \setminus Z$. For the first sum:

$$\sum_{z \in Z} |\rho_{\sigma',d'}(z) - \rho_{\sigma,d}(z)| \leq 2\delta_3 \sum_{z \in Z} |\rho_{\sigma,d}(z)| \leq 2\delta_3(1 + 1.01 \cdot 10^{-8}) \approx 2\delta_3$$

Now, for the second sum, Lemma 4.4, part 1, from [Lyu12] states that⁸:

$$\text{For any } k > 0, \mathbb{P}[|z - d| > k\sigma; z \leftarrow D_{\mathbb{Z},\sigma,d}] \leq 2e^{-k^2/2}$$

Using this lemma, it is straightforward that

$$\begin{aligned} \sum_{z \in \mathbb{Z} \setminus Z} |\rho_{\sigma,d}(z) - \rho_{\sigma,d'}(z)| &\leq \sum_{z \in \mathbb{Z} \setminus Z} \rho_{\sigma,d}(z) + \rho_{\sigma,d}(z) \\ &\leq 4e^{-k^2/2} (\rho_{\sigma,c}(\mathbb{Z}) + \rho_{\sigma',c'}(\mathbb{Z})) \\ &\leq 8e^{-k^2/2} \delta_3 (1 + 1.01 \cdot 10^{-8}) \approx 8e^{-k^2/2} \end{aligned}$$

□

⁸ It is actually stated only for $d = 0$, but the proofs holds $\forall d \in \mathbb{R}$.

Lemma 10 (Bounded ratio of discrete gaussians over a finite set). Let $\sigma, \sigma' > 1$ and $d, d' \in (-1/2, 1/2]$. Let $k > 0$ and $z \in \mathbb{Z}$ such that $|z - d| \leq k\sigma$. Suppose $\exists \delta_3 \in (0, 1/2)$ such that $1 - 2\delta_3 \leq \frac{\rho_{\sigma', d'}(z)}{\rho_{\sigma, d}(z)} \leq 1 + 2\delta_3$. Then:

$$\left| \frac{D_{\mathbb{Z}, \sigma', c'}(z)}{D_{\mathbb{Z}, \sigma, c}(z)} - 1 \right| \leq \delta_4$$

where $\delta_4 = 4\delta_3 + 4e^{-k^2/2} + 4\delta_3^2 + 8\delta_3 e^{-k^2/2}$. In practice δ_3 is small and k is “somewhat” big, so $\delta_4 \approx 4\delta_3 + 4e^{-k^2/2}$.

Proof.

$$\begin{aligned} |D_{\mathbb{Z}, \sigma, c}(z) - D_{\mathbb{Z}, \sigma', c'}(z)| &= \left| \frac{\rho_{\sigma, c}(z)}{\rho_{\sigma, c}(\mathbb{Z})} - \frac{\rho_{\sigma', c'}(z)}{\rho_{\sigma', c'}(\mathbb{Z})} \right| \\ &= \frac{\rho_{\sigma, c}(z)}{\rho_{\sigma, c}(\mathbb{Z})} \left| 1 - \frac{\rho_{\sigma', c'}(z)}{\rho_{\sigma, c}(z)} \times \frac{\rho_{\sigma, c}(\mathbb{Z})}{\rho_{\sigma', c'}(\mathbb{Z})} \right| \\ &\leq \frac{\rho_{\sigma, c}(z)}{\rho_{\sigma, c}(\mathbb{Z})} \left| 1 - (1 + 2\delta_3)(1 + 2\delta_3 + 4e^{-k^2/2}) \right| \\ &\leq D_{\mathbb{Z}, \sigma, c}(z) \left(4\delta_3 + 4e^{-k^2/2} + 4\delta_3^2 + 8\delta_3 e^{-k^2/2} \right) \end{aligned}$$

where the penultimate line is obtained by bounding $\frac{\rho_{\sigma, c}(\mathbb{Z})}{\rho_{\sigma', c'}(\mathbb{Z})}$ via Lemma 9. \square

References

- [ABB10] Agrawal, S., Boneh, D., Boyen, X.: Lattice Basis Delegation in Fixed Dimension and Shorter-Ciphertext Hierarchical IBE. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 98–115. Springer, Heidelberg (2010)
- [Bab86] Babai, L.: On lovász’ lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986)
- [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)
- [Boy10] Boyen, X.: Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In: Public Key Cryptography, pp. 499–517 (2010)
- [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)
- [DDLL13] Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice Signatures and Bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013)
- [DLP14] Ducas, L., Lyubashevsky, V., Prest, T.: Efficient Identity-Based Encryption over NTRU Lattices. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 22–41. Springer, Heidelberg (2014)
- [Gal12] Galbraith, S.D.: Mathematics of Public Key Cryptography, 1st edn. Cambridge University Press, New York (2012)

- [GHN06] Gama, N., Howgrave-Graham, N., Nguyn, P.Q.: Symplectic Lattice Reduction and NTRU. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 233–253. Springer, Heidelberg (2006)
- [GPV08] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC, pp. 197–206 (2008)
- [Hig02] Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2002)
- [HPS98] 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)
- [Kle00] Klein, P.N.: Finding the closest lattice vector when it’s unusually close. In: SODA, pp. 937–941 (2000)
- [LLL82] Lenstra, A.K., Lenstra, H.W., Lovsz, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**, 515–534 (1982)
- [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)
- [LMPR08] Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: A Modest Proposal for FFT Hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer, Heidelberg (2008)
- [LPR13a] 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)
- [LPR13b] 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)
- [LTv12] Lpez-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: STOC, pp. 1219–1234 (2012)
- [Lyu12] Lyubashevsky, V.: Lattice Signatures without Trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012)
- [MS07] Vladimir Maz’ya and Gunther Schmidt. Approximate Approximations. AMS, 1st edn. (2007)
- [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)
- [SSTX09] Stehl , 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)
- [Swe84] Sweet, D.R.: Fast toeplitz orthogonalization. Numerische Mathematik **43**, 1–21 (1984)

Author Index

- Abdalla, Michel II-69
Abdelraheem, Mohamed Ahmed I-762
Abe, Masayuki II-35
Afshar, Arash I-702
Agrawal, Shashank II-501
Agrawal, Shweta II-501
Albrecht, Martin R. I-430
Asharov, Gilad I-673

Balasch, Josep I-486
Barbulescu, Razvan I-129
Bar-On, Achiya I-315
Barthe, Gilles I-457, II-689
Beelen, Peter I-762
Belaïd, Sonia I-457
Bellare, Mihir II-627
Benhamouda, Fabrice II-69
Bernstein, Daniel J. I-368
Bogdanov, Andrey I-762
Boneh, Dan II-404, II-563
Boyle, Elette II-337

Canteaut, Anne I-45
Chase, Melissa II-532
Chen, Jie II-595
Cheon, Jung Hee I-3, I-513
Chiesa, Alessandro II-371
Cogliati, Benoît I-584
Cramer, Ronald II-313
Crespo, Juan Manuel II-689

Dachman-Soled, Dana II-131
Dagdelen, Özgür II-719
Damgård, Ivan Bjerre II-313
Ding, Jintai II-719
Dinur, Itai I-231, I-315, I-733
Dodis, Yevgeniy I-101
Döttling, Nico II-313
Duc, Alexandre I-173, I-401
Ducas, Léo I-617
Dunkelman, Orr I-315
Dupressoir, François I-457
Dziembowski, Stefan II-159

Evans, David II-220
Faust, Sebastian I-401, I-486, II-159
Fehr, Serge II-313
Fouque, Pierre-Alain I-457
Frederiksen, Tore Kasper II-191

Ganesh, Chaya I-101
Garay, Juan II-281
Gaudry, Pierrick I-129
Gay, Romain II-595
Gierlichs, Benedikt I-486
Gilboa, Niv II-337
Golovnev, Alexander I-101
Grégoire, Benjamin I-457
Groth, Jens II-253
Guillevic, Aurore I-129
Gupta, Divya II-404

Halevi, Shai I-641
Han, Kyohyung I-3
Hoang, Viet Tung I-15, II-627
Hohenberger, Susan II-3
Hopwood, Daira I-368
Hu, Zhangxiang I-702
Hülsing, Andreas I-368

Ishai, Yuval II-337

Jia, Dingding I-559
Juels, Ari I-101

Keller, Nathan I-315
Kiayias, Aggelos II-281, II-468
Kiltz, Eike II-101
Kohlweiss, Markulf II-35, II-253
Koppula, Venkata II-3
Krovetz, Ted I-15
Kurosawa, Kaoru I-537

Lakhnech, Yassine II-689
Lallemand, Virginie I-315

- Lange, Tanja I-368
 Leander, Gregor I-254
 Lee, Changmin I-3
 Leonardos, Nikos II-281
 Leurent, Gaëtan I-345
 Lewi, Kevin II-563
 Li, Bao I-559
 Lindell, Yehuda I-673
 Liu, Feng-Hao II-131
 Liu, Yi-Kai II-785
 Lu, Xianhui I-559
 Lyubashevsky, Vadim I-789
- May, Alexander I-203
 Micciancio, Daniele I-617
 Minaud, Brice I-254
 Mironov, Ilya II-404, II-657
 Mohassel, Payman I-702
 Morain, François I-129
 Morawiecki, Paweł I-733
- Niederhagen, Ruben I-368
 Nielsen, Jesper Buus II-191
 Nuida, Koji I-537
- Ohkubo, Miyako II-35
 Orlandi, Claudio II-191
 Ostrovsky, Rafail II-532
 Ozerov, Ilya I-203
- Papachristodoulou, Louiza I-368
 Pieprzyk, Josef I-733
 Pierrot, Cécile I-156
 Pointcheval, David II-69
 Prabhakaran, Manoj II-501
 Prest, Thomas I-789
- Raykova, Mariana II-563
 Rechberger, Christian I-430
 Ristenpart, Thomas I-101
 Rogaway, Phillip I-15
 Rønjom, Sondre I-254
 Rosulek, Mike I-702, II-220
 Roué, Joëlle I-45
 Ryu, Hansol I-3
- Sahai, Amit II-404, II-563
 Schmidt, Benedikt II-689
- Schneider, Michael I-368
 Schneider, Thomas I-430, I-673
 Schwabe, Peter I-368
 Seurin, Yannick I-584
 Shoup, Victor I-641
 Shrimpton, Thomas I-77
 Skorski, Maciej II-159
 Snook, Michael II-719
 Spini, Gabriele II-313
 Srebrny, Marian I-733
 Standaert, François-Xavier I-401
 Stehlé, Damien I-3, I-513
 Stephens-Davidowitz, Noah II-657
 Straus, Michał I-733
 Strub, Pierre-Yves I-457
- Terashima, R. Seth I-77
 Tibouchi, Mehdi II-35
 Tiessen, Tyge I-430
 Tischhauser, Elmar I-762
 Todo, Yosuke I-287
 Tramèr, Florian I-173
 Tromer, Eran II-371
 Tsaban, Boaz I-315
- Unruh, Dominique II-755
- Vaudenay, Serge I-173
 Virza, Madars II-371
 Visconti, Ivan II-532
- Wang, Lei I-345
 Waters, Brent II-3
 Wee, Hoeteck II-101, II-595
 Wilcox-O'Hearn, Zooko I-368
- Yun, Aaram II-817
- Zacharias, Thomas II-468
 Zahur, Samee II-220
 Zhandry, Mark II-563
 Zhang, Bingsheng II-468
 Zhang, Jiang II-719
 Zhang, Zhenfeng II-719
 Zhou, Hong-Sheng II-131
 Zimmerman, Joe II-439, II-563
 Zohner, Michael I-430, I-673