Alejandro Hevia
Gregory Neven (Eds.)

LNCS 7533

Progress in Cryptology – LATINCRYPT 2012

2nd International Conference on Cryptology and Information Security in Latin America
Santiago, Chile, October 2012, Proceedings

Springer

Lecture Notes in Computer Science 7533

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Alejandro Hevia Gregory Neven (Eds.)

Progress in Cryptology – LATINCRYPT 2012

2nd International Conference on Cryptology and Information Security in Latin America
Santiago, Chile, October 7-10, 2012
Proceedings

Springer

Volume Editors

Alejandro Hevia
University of Chile
Department of Computer Science
Blanco Encalada 2120
Tercer Piso, Santiago, Chile
E-mail: ahevia@dcc.uchile.cl

Gregory Neven
IBM Research - Zurich
Säumerstrasse 4
8803 Rüschlikon, Switzerland
E-mail: nev@zurich.ibm.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-33480-1 e-ISBN 978-3-642-33481-8
DOI 10.1007/978-3-642-33481-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012946763

CR Subject Classification (1998): E.3, K.6.5, D.4.6, C.2, G.2, E.4

LNCS Sublibrary: SL 4 – Security and Cryptology

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Latincrypt 2012 was the Second International Conference on Cryptology and Information Security in Latin America and took place from October 7th to 10th, 2012, in Santiago, Chile. The conference was organized by the Group of Applied Cryptography and Security (CLCERT) of the University of Chile and the NIC Chile Research Labs in cooperation with The International Association for Cryptologic Research (IACR). The General Chairs of the conference were Javier Bustos and Alejandro Hevia.

By the submission deadline on May 18th, 2012, the conference had received 47 submissions. Each submission was reviewed by at least three committee members, submissions co-authored by members of the Program Committee were assigned to at least five committee members. The reviewing process was challenging due to the high quality of the submissions, and we are deeply grateful to the committee members and external reviewers for their outstanding work. After meticulous deliberation, the Program Committee, which was chaired by Alejandro Hevia and Gregory Neven, selected 17 submissions for presentation at the conference. These are the articles included in this volume. In addition to these presentations, the program also included four invited talks and a student poster session.

The reviewing process was run using the iChair software written by Thomas Baignères and Matthieu Finiasz. We are especially grateful to them for letting us use their software and for their prompt responses to our questions on how to use the system. Also, we thank Sergio Miranda from the CLCERT for his help in setting up the reviewing system.

Finally, we would like to thank our sponsors Intel & McAfee Chile, NIC Chile, Certivox, the Center for Mathematical Modeling (CMM) of the University of Chile, INRIA Chile, CLEI, Yahoo Research Chile, and Orand Chile, for their financial support, as well as all the people who contributed to the success of this conference. In particular, we are indebted to the members of the Latincrypt Steering Committee, especially to Michel Abdalla, who we regularly consulted for his experience from Latincrypt 2010; the General Co-chair Javier Bustos; Jacqueline Araya; and everyone in the Local Organizing Committee for their diligent work and for making this conference possible. Finally, we would like to thank Springer for publishing the proceedings in their *Lecture Notes in Computer Science* series.

October 2012

Alejandro Hevia
Gregory Neven

LATINCRYPT 2012

Second International Conference on
Cryptology and Information Security in Latin America

Santiago, Chile
October 7–10, 2012

Organized by
Grupo de Criptografía Aplicada y Seguridad (CLCERT), NIC Chile Research Labs & Dept. of Computer Science, Universidad de Chile

In Cooperation with
The International Association for Cryptologic Research (IACR)

General Chairs

Javier Bustos	NIC Chile Research Labs, Chile
Alejandro Hevia	CLCERT & Department of Computer Science, Universidad de Chile, Chile

Program Chairs

Alejandro Hevia	CLCERT & Department of Computer Science, Universidad de Chile, Chile
Gregory Neven	IBM Research – Zurich, Switzerland

Steering Committee

Michel Abdalla	École Normale Supérieure, France
Paulo Barreto	Universidade de São Paulo, Brazil
Ricardo Dahab	Universidade Estadual de Campinas, Brazil
Alejandro Hevia	Universidad de Chile, Chile
Julio López	Universidade Estadual de Campinas, Brazil
Daniel Panario	Carleton University, Canada
Alfredo Viola	Universidad de la República, Uruguay

Local Organizing Committee

Jacqueline Araya	NIC Chile Research Labs, Chile
Sergio Miranda	CLCERT, Chile
Alonso González	Universidad de Chile, Chile
Philippe Camacho	Universidad de Chile, Chile
Rodrigo Abarzúa	Universidade Estadual de Campinas, Brazil

Program Committee

Michel Abdalla	École Normale Supérieure, France
Roberto Avanzi	Ruhr-University Bochum, Germany
Paulo Barreto	University of São Paulo, Brazil
Lejla Batina	Radboud University Nijmegen, The Netherlands
Philippe Camacho	Universidad de Chile, Chile
Claude Carlet	Université Paris 8, France
Carlos Cid	Royal Holloway, University of London, UK
Ricardo Dahab	Universidade Estadual de Campinas, Brazil
Joan Daemen	ST Microelectronics, Belgium
Orr Dunkelmann	University of Haifa, Israel
Stefan Dziembowski	University of Warsaw, Poland, and University of Rome "La Sapienza", Italy
Sebastian Faust	Aarhus University, Denmark
Georg Fuchsbauer	University of Bristol, UK
Philippe Gaborit	Université de Limoges, France
Joachim von zur Gathen	B-IT, Universität Bonn, Germany
Tibor Jager	Karlsruhe Institute of Technology, Germany
Seny Kamara	Microsoft Research, USA
Stefan Katzenbeisser	Technische Universität Darmstadt, Germany
Vladimir Kolesnikov	Bell Labs, USA
Sven Laur	University of Tartu, Estonia
Vadim Lyubashevsky	École Normale Supérieure, France
Daniel Panario	Carleton University, Canada
Giuseppe Persiano	Università di Salerno, Italy
Carla Ràfols	Ruhr-Universität Bochum, Germany
Christian Rechberger	DTU, Denmark
Tamara Rezk	INRIA Sophia Antipolis-Méditerranée, France
Matt Robshaw	Orange Labs, France
Francisco Rodríguez-Henríquez	Centro de Investigación y de Estudios Avanzados del I.P.N., México
Nicolas Thériault	Universidad del Bío-Bío, Chile
Maribel Gonzalez Vasco	Universidad Rey Juan Carlos, Spain
Alfredo Viola	Universidad de la República, Uruguay
Ivan Visconti	Università di Salerno, Italy
Scott Yilek	University of St. Thomas, USA
Santiago Zanella–Béguelin	Microsoft Research, UK

External Reviewers

Mohamed Ahmed Abdelraheem
Diego Aranha
Raoul Blankertz
Anne Canteaut
Julio Cesar Hernández Castro
Angelo De Caro
Chun-I Fan
Junfeng Fan
Philippe Gaborit
Steven Galbraith
Alonso González
Shay Gueron
Iftach Haitner
Darrel Hankerson
Jennie Hansen
Clemens Heuberger
Aggelos Kiayias
Markulf Kohlweiss
Daniel Loebenberger
Zhengqin Luo
Gilles Macario-Rat
Wilfried Meidl
Marine Minier
Michael Naehrig
Michael Nüsken
Roger Oyono
Valerio Pastro
Geovandro Pereira
Christiane Peters
Jeremy Planul
Alex Pott
Jordi Pujolàs
Guénaël Renault
Yannis Rouselakis
Peter Schwabe
Christoph Striecks
Enrico Thomae
Viet-Cuong Trinh
Marcin Wojcik
Konstantin Ziegler

Sponsoring Institutions

Grupo de Criptografía Aplicada y Seguridad (CLCERT), Universidad de Chile
NIC Chile Research Labs, Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile
Intel & McAfee Labs
Center for Mathematical Modeling, Universidad de Chile
Certivox
NIC Chile
Centro Latinoamericano de Estudios en Informáticas (CLEI)
INRIA Chile
Yahoo! Labs
Orand
Welcu

Table of Contents

Elliptic Curves

Cryptographic Protocols I

Implementations

Cryptographic protocols II

Foundations

Symmetric-key Cryptography

Indifferentiable Hashing to Barreto–Naehrig Curves

Pierre-Alain Fouque[1] and Mehdi Tibouchi[2]

[1] École Normale Supérieure and INRIA Rennes
pierre-alain.fouque@ens.fr
[2] NTT Secure Platform Laboratories
tibouchi.mehdi@lab.ntt.co.jp

Abstract. A number of recent works have considered the problem of constructing constant-time hash functions to various families of elliptic curves over finite fields. In the relevant literature, it has been occasionally asserted that constant-time hashing to certain special elliptic curves, in particular so-called BN elliptic curves, was an open problem. It turns out, however, that a suitably general encoding function was constructed by Shallue and van de Woestijne back in 2006.

In this paper, we show that, by specializing the construction of Shallue and van de Woestijne to BN curves, one obtains an encoding function that can be implemented rather efficiently and securely, that reaches about 9/16ths of all points on the curve, and that is *well-distributed* in the sense of Farashahi *et al.*, so that one can easily build from it a hash function that is indifferentiable from a random oracle.

Keywords: Elliptic curve cryptography, BN curves, hashing, random oracle.

1 Introduction

Many elliptic curve-based cryptographic protocols require hashing to an elliptic curve group $\mathbb{G}$: they involve one or more hash functions $\mathfrak{H} : \{0,1\}^* \to \mathbb{G}$ mapping arbitrary values to points on the elliptic curve.

For example, in the Boneh-Franklin identity-based encryption scheme [7], the public key for identity $\mathrm{id} \in \{0,1\}^*$ is a point $\mathbf{Q}_{\mathrm{id}} = \mathfrak{H}(\mathrm{id})$ on the curve. This is also the case in many other pairing-based cryptosystems including IBE and HIBE schemes [1,24,25], signature and identity-based signature schemes [6,8,9,14,42] and identity-based signcryption schemes [11,32].

Hashing into elliptic curves is also required for some passwords-based authentication protocols such as the SPEKE [27] and PAK [12] protocols, as well as various signature schemes based on the hardness of the discrete logarithm problem, like [15], when they are instantiated over elliptic curves.

In all of those cases, the hash functions are modeled as random oracles in security proofs. However, it is not clear how such a hash function can be instantiated in practice. Indeed, random oracles to groups like $\mathbb{Z}_p^*$ can be easily

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 1–17, 2012.

constructed from random oracles to fixed-length bit strings, for which conventional cryptographic hash functions usually provide acceptable substitutes. On the other hand, constructing random oracles to an elliptic curve even from random oracles to bit strings appears difficult in general, and some of the more obvious instantiations actually break security completely.

For example, to construct a hash function $\mathfrak{H} : \{0,1\}^* \to \mathbb{G}$ to an elliptic curve cyclic group $\mathbb{G}$ of order N and generated by a given point $\mathbf{G}$, a simple idea might be to choose $\mathfrak{H}(m) = [\mathfrak{h}(m)] \cdot \mathbf{G}$, where $\mathfrak{h}$ is a hash function $\{0,1\}^* \to \mathbb{Z}_N$. However, this typically breaks security proofs in the random oracle model. Suppose the proof involves programming the random oracle $\mathfrak{H}$ by choosing its value $\mathbf{P}$ on some input m_0. If we instantiate $\mathfrak{H}$ in this naive way, the programming stage requires setting $\mathfrak{h}(m_0)$ to the discrete logarithm of $\mathbf{P}$, which is normally unknown. In fact, in the case of a primitive like BLS signatures [10], this instantiation leads to very simple and devastating attacks (see the discussions in [40, Ch. 3] or [41]).

In their original short signatures paper [9], Boneh, Lynn and Shacham introduced the first generic construction of a secure hash function to elliptic curves, in the sense that it applies to any target elliptic curve: the so-called "try-and-increment" algorithm. Basically, to hash a message m, one concatenates it with a fixed-length counter c initialized to 0 and computes $\mathfrak{h}(c\|m)$, where $\mathfrak{h}$ is a hash function to the base field of the elliptic curve. If that digest value is the abscissa of a point on the curve, $\mathfrak{H}(m)$ is set to that point; otherwise, one increments the counter c and tries again. This construction can be shown to be secure provided that the counter length is large enough; however, it is somewhat inefficient, since one may need many iterations before finding a suitable point, and the fact that the length of the computation depends on the input yields to possible side-channel attacks, especially in protocols such as password authenticated key exchange (a concrete attack is given is [41] for a variant of SPEKE [27]).

In response, more robust, "constant-time" methods have been proposed, starting with a paper by Icart at CRYPTO 2009 [26], and including a number of extensions, generalizations and refinements afterwards [36,13,21,30,18,16]. In essence, these methods are all based on the construction of a suitable algebraic or piecewise algebraic function from the affine line to the target elliptic curve.

Our Contributions. None of the methods mentioned above is fully generic: they all rely on certain arithmetic or geometric properties of the target curve. Some involve taking arbitrary cube roots in the base field $\mathbb{F}_q$, for example, and hence only apply to the case when $q \equiv 2 \pmod 3$. The remaining ones only work for curves of nonzero j-invariant. In particular, none of those more efficient encodings yield a construction of hash functions to the very important class of Barreto–Naehrig (BN) elliptic curves [4], which are the preferred curves for implementing asymmetric pairings nowadays, as they provide essentially optimal parameters for the 128-bit security level. This has led several authors to assert that constant-time hashing to BN curves was an open problem [21,34].

It turns out, however, that several years prior to Icart's work, Shallue and van de Woestijne had presented a construction [38] that applies to all curves of odd characteristic, as pointed out in [41]. This paper is devoted to making that

construction explicit in the case of BN curves, and establishing some properties of it. More precisely, our contributions are as follows:

- we propose an explicit, optimized definition of the Shallue–van de Woestijne encoding to a BN elliptic curve (in Section 3);
- using an extension of the technique from [20,22], we establish an estimate for the number of points in the image of that encoding (in Section 4): we find that the encoding reaches about 9/16ths of all points on the curve;
- like Icart's encoding and many others, this encoding f will not yield a generically secure hash function construction if we simply compose it with a random oracle to the base field (e.g. it is easy to distinguish such a hash function from a random oracle to the curve since its image has a simple algebraic description and only contains a constant fraction of all points). However, we show (in Section 5) that it is *well-distributed* in the sense of Farashahi *et al.* [19]. This implies that if $\mathfrak{h}_1$, $\mathfrak{h}_2$ are random oracles to the base field, then $m \mapsto f\big(\mathfrak{h}_1(m)\big) + f\big(\mathfrak{h}_2(m)\big)$ is a good, generically secure hash function to the BN curve (it is *indifferentiable* from a random oracle);
- finally, we also suggest (in Section 6) a way to implement this encoding function that should thwart side-channel analysis and other physical attacks.

Our approach to establishing the results of Sections 4 and 5, while quite technical, is also of independent interest. Indeed, the Shallue–van de Woestijne encoding fits in a family of various encoding functions to elliptic curves based on works by Schinzel and Skałba [37,39], and while Fouque and Tibouchi [22] and Farashahi *et al.* [19] did tackle a function of that type before, they had to consider only a special case and tweak the formulas significantly, so as to simplify the computations. In this paper, we show how image size estimates and well-distributedness can be obtained for this type of encoding functions without simplifications or generality loss.

Our results apply almost without change to any elliptic curve of the form $y^2 = x^3 + b$ with $b \neq -1$ over a finite field $\mathbb{F}_q$ with $q \equiv 7 \pmod{12}$. Pairing-friendly curves obtained by the CM method for discriminant -3 are typically of that form: this includes in particular the curves constructed by Barreto, Lynn and Scott in [3, §3.1] and the curves of embedding degree 18 and 36 obtained by Kachisa, Schaefer and Scott in [29], all of which are recommended for pairing implementations at higher security levels (192 to 256-bit security). The elliptic curve group in those cases is not usually of prime order, however (those cases have $\rho > 1$), so hashing to the prime order subgroup requires multiplying the point obtained with the technique described herein by the cofactor. This does not affect indifferentiability, as was shown in [13, §6.1].

Notation. In the paper, p will always be an odd prime and q an odd prime power. In $\mathbb{F}_q$, the finite field with q elements, we denote by $\chi_q \colon \mathbb{F}_q \to \{-1, 0, 1\}$ the nontrivial quadratic character of $\mathbb{F}_q^*$ extended by zero to $\mathbb{F}_q$ (i.e. $\chi_q(0) = 0$ and for $a \neq 0$, $\chi_q(a) = 1$ if a is a square and -1 otherwise). When $q \equiv 3 \pmod{4}$, we write $\sqrt{a} = a^{(q+1)/4}$ for any square $a \in \mathbb{F}_q$.

2 Preliminaries

2.1 Barreto–Naehrig Elliptic Curves

BN curves are a family of pairing-friendly elliptic curves over large prime fields, introduced in 2005 by Barreto and Naehrig [4]. They are one of the preferred families for implementing asymmetric pairings nowadays, as they achieve essentially optimal parameters for obtaining bilinear groups at the 128-bit security level. Indeed, BN curves are of prime order (in particular they satisfy $\rho = 1$) and embedding degree $k = 12$; thus, the pairing on a BN curve over a 256-bit prime field $\mathbb{F}_p$ takes its values in the field $\mathbb{F}_{p^{12}}$ of size $256 \times 12 = 3072$. Then, solving the discrete logarithm problem both in the group of points of the curve and in $\mathbb{F}_{p^k}^{\times}$ takes time about 2^{128} as required [4].

The details of the construction of BN curves, based on the CM method, is not really relevant for our purposes. Suffice it to say that Barreto and Naehrig's algorithm outputs an elliptic curve of the form:

$$E\colon y^2 = x^3 + b \tag{1}$$

over a field $\mathbb{F}_p$ with $p \equiv 1 \pmod 3$ (for convenience, they suggest to pick a p satisfying, more precisely, $p \equiv 31 \pmod{36}$), such that $\#E(\mathbb{F}_p)$ is prime, together with the generator[1] $\mathbf{G} = (1, \sqrt{b+1} \bmod p) \in E(\mathbb{F}_p)$. Moreover, b is typically a very small integer (the smallest > 0 such that $b + 1$ is a quadratic residue mod p).

2.2 Chebotarev Density Theorem

In [20,22], Farashahi, Shparlinski and Voloch on the one hand, and Fouque and Tibouchi on the other, proposed an approach to counting the number of points in the image of an elliptic curve encoding function, based on the Chebotarev density theorem for function fields. We will apply a similar technique to the encoding to BN curves presented hereafter, and will therefore need an effective version of the Chebotarev density theorem. One such version is given in [23, Proposition 6.4.8], and if we specialize it to our cases of interest, we obtain:

Lemma 1 (Chebotarev). *Let K be an extension of $\mathbb{F}_q(x)$ of degree $d < \infty$ and L a Galois extension of K of degree $m < \infty$. Assume $\mathbb{F}_q$ is algebraically closed in L, and fix some subset $\mathscr{S}$ of $\mathrm{Gal}(L/K)$ stable under conjugation. Let $s = \#\mathscr{S}$ and $N(\mathscr{S})$ the number of places v of K of degree 1, unramified in L, such that the Artin symbol $\left(\frac{L/K}{v}\right)$ (defined up to conjugation) is in $\mathscr{S}$. Then*

$$\left| N(\mathscr{S}) - \frac{s}{m} q \right| \leq \frac{2s}{m} \left((m + g_L) \cdot q^{1/2} + m(2g_K + 1) \cdot q^{1/4} + g_L + dm \right)$$

where g_K and g_L are the genera of the function fields K and L.

[1] Later works such as [34] use a different point as the generator, and the corresponding construction does no longer ensure that $1 + b$ is a square. This only causes a minor inconvenience for our purposes, namely two extra elements of $\mathbb{F}_q$ that have to be treated separately in the encoding given in Section 3.

2.3 Admissible Encodings and Indifferentiability

Brier *et al.* [13] use Maurer's indifferentiability framework [33] to analyze the conditions under which their hash function constructions can be plugged into essentially any scheme[2] that is proved secure in the random oracle model in such a way that the proof of security goes through. As shown by Maurer, it suffices that the hash function construction be indifferentiable from a random oracle.

Then, Brier *et al.* [13] establish a sufficient condition for a hash function construction into an elliptic curve E to be indifferentiable from a random oracle. It applies to hash functions of the form:

$$\mathfrak{H}(m) = F(\mathfrak{h}(m)),$$

where $F\colon S \to E(\mathbb{F}_q)$ is a deterministic encoding, and $\mathfrak{h}$ is seen as a random oracle to S. Assuming that $\mathfrak{h}$ is a random oracle, the construction is indifferentiable whenever F is an *admissible encoding* into $E(\mathbb{F}_q)$, in the sense that it satisfies the following properties:

1. Computable: F is computable in deterministic polynomial time;
2. Regular: for s uniformly distributed in S, the distribution of $F(s)$ is statistically indistinguishable from the uniform distribution in $E(\mathbb{F}_q)$;
3. Samplable: there is an efficient randomized algorithm $\mathscr{I}$ such that for any $\mathbf{P} \in E(\mathbb{F}_q)$, the distribution of $\mathscr{I}(\mathbf{P})$ is statistically indistinguishable from the uniform distribution in $F^{-1}(\mathbf{P})$.

2.4 Well-Distributed Elliptic Curve Encodings

Building upon this work by Brier *et al.*, Farashahi *et al.* introduced a general framework [19] to obtain well-behaved hash function construction to elliptic and hyperelliptic curves. The main notion in that framework is that of a *well-distributed* encoding. In the case of an elliptic curve E, it is defined as follows.

Definition 1 (Farashahi *et al.*). *A function $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$ is said to be B-well-distributed for some $B > 0$ if, for all nontrivial characters χ of $E(\mathbb{F}_q)$, the following bound holds:*

$$|S_f(\chi)| \leq B\sqrt{q}, \quad \textit{where} \quad S_f(\chi) = \sum_{u \in \mathbb{F}_q} \chi\big(f(u)\big). \tag{2}$$

[2] It has recently been pointed out by Ristenpart, Shacham and Shrimpton [35] that this type of composition result does not apply to literally *all* cryptographic protocols, but only those which admit so-called "single-stage security proofs". This is not a significant restriction for the purpose at hand, as all protocols constructed so far using elliptic curve-valued hashing satisfy that requirement.

Let $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$ be a well-distributed encoding to the elliptic curve E. Then Farashahi *et al.* prove that the tensor square $f^{\otimes 2}\colon \mathbb{F}_q^2 \to E(\mathbb{F}_q)$ defined by $f^{\otimes 2}(u,v) = f(u) + f(v)$ is a *regular* encoding to $E(\mathbb{F}_q)$. More precisely, the statistical distance between the distribution defined by $f^{\otimes 2}$ on $E(\mathbb{F}_q)$ and the uniform distribution is bounded above by $B^2 \cdot \sqrt{\#E(\mathbb{F}_q)}/q$, which is negligible.

If the function f is also efficiently computable and samplable in the sense of Section 2.3, then $f^{\otimes 2}$ is *admissible*, which implies that the hash function:

$$m \mapsto f\big(\mathfrak{h}_1(m)\big) + f\big(\mathfrak{h}_2(m)\big)$$

is indifferentiable from a random oracle, and hence can be used in lieu of a random oracle to $E(\mathbb{F}_q)$ in essentially any scheme proved secure in the random oracle model.

Another important result of [19] is the following consequence of the Riemann hypothesis for curves, which makes it possible to establish the bound (2) in practice.

Lemma 2 ([19, Th. 7]). *Let $h : X \to E$ be a non constant morphism to the elliptic curve E, and χ be any nontrivial character of $E(\mathbb{F}_q)$. Assume that h does not factor through a nontrivial unramified morphism $Z \to E$. Then:*

$$\left| \sum_{\mathbf{P} \in X(\mathbb{F}_q)} \chi(h(P)) \right| \leq (2g_X - 2)\sqrt{q} \tag{3}$$

where g_X is the genus of X. Furthermore, if φ is a non constant rational function on X:

$$\left| \sum_{\mathbf{P} \in X(\mathbb{F}_q)} \chi(h(P)) \cdot \chi_q(\varphi(\mathbf{P})) \right| \leq (2g_X - 2 + 2\deg\varphi)\sqrt{q}. \tag{4}$$

2.5 The Shallue–van de Woestijne Encoding

Let E be any elliptic curve over a finite field $\mathbb{F}_q$ of odd characteristic with $\#\mathbb{F}_q > 5$, written is Weierstrass form:

$$E\colon y^2 = g(x) = x^3 + Ax^2 + Bx + C.$$

Shallue and van de Woestijne [38] construct an encoding function $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$ as follows.

Following the ideas of Schinzel and Skałba [37,39], they first introduce the algebraic threefold $V \subset \mathbb{P}^4$ with affine equation

$$V\colon y^2 = g(x_1) \cdot g(x_2) \cdot g(x_3),$$

and observe that if (x_1, x_2, x_3, y) is an $\mathbb{F}_q$-rational point on V, then at least one of x_1, x_2, x_3 is the abscissa of a point in $E(\mathbb{F}_q)$. Indeed, the product $g(x_1) \cdot g(x_2) \cdot g(x_3) \in \mathbb{F}_q$ is a square, so at least one of the factors must be square as well. Then, they establish the following result.

Lemma 3 ([38, Lemma 6]). *Put* $h(u,v) = u^2 + uv + v^2 + A(u+v) + B$, *and define:*

$$S\colon y^2 \cdot h(u,v) = -g(u),$$
$$\psi\colon (u,v,y) \mapsto \left(v, -A-u-v, u+y^2, g(u+y^2)\cdot h(u,v)\cdot y^{-1}\right).$$

Then ψ *is a rational map from the surface* S *to* V *that is invertible on its image.*

In particular, any point in $S(\mathbb{F}_q)$ where ψ is well-defined (i.e. satisfying $y \neq 0$) maps to a point in $V(\mathbb{F}_q)$, and hence yields a point in $E(\mathbb{F}_q)$. Finally, to construct points on S, the authors of [38] note that any plane section of S of the form $u = u_0$ is birational to a conic, which is non-degenerate as long as:

$$g(u_0) \neq 0 \quad \text{and} \quad 3u_0^2 + 2Au_0 + 4B - A^2 \neq 0. \tag{5}$$

If we fix one such value u_0 (it necessarily exists since $\#\mathbb{F}_q > 5$), the corresponding conic admits a rational parametrization, which gives a rational map $\phi\colon \mathbb{A}^1 \to S$.

The encoding function $f\colon \mathbb{F}_q \to E(\mathbb{F}_q)$ is then obtained as mapping a point $t \in \mathbb{F}_q$ to one of the points on $E(\mathbb{F}_q)$ of abscissa x_i, where $\psi \circ \phi(t) = (x_1, x_2, x_3, y)$ and $i \in \{1,2,3\}$ is the smallest index such that $g(x_i)$ is a square.

In the following sections, we make that function f explicit when E is a BN curve as in Section 2.1 (or rather, belongs to a class of elliptic curves that contains BN curves), and establish a number of its properties.

3 An Encoding to BN Curves

Let us apply the previous construction to the case of an elliptic curve of the form:

$$E\colon y^2 = g(x) = x^3 + b$$

over a field $\mathbb{F}_q$ of characteristic ≥ 5. We also assume that $q \equiv 7 \pmod{12}$ and that $g(1) = 1 + b$ is a nonzero square in $\mathbb{F}_q$. As seen in Section 2.1, all those properties are in particular satisfied for BN curves.[3]

The equation of the surface S defined in Section 2.5 becomes:

$$S\colon y^2 \cdot (u^2 + uv + v^2) = -u^3 - b.$$

We consider its section by the plane of equation $u = u_0 = 1$. This gives a curve of equation:

$$y^2 \cdot \left(\frac{3}{4} + \left(v + \frac{1}{2}\right)^2\right) = -1 - b,$$

and by setting $z = v + 1/2$ and $w = 1/y$, we see that it is birational to the conic:

$$z^2 + (1+b)w^2 = -\frac{3}{4}, \tag{6}$$

[3] Technically, one can consider BN curves over fields $\mathbb{F}_p$ with $p \equiv 1 \pmod{12}$ as well, but they are usually avoided in practice, as the condition $p \equiv 3 \pmod 4$ makes square roots more convenient to compute.

which is non-degenerate since $g(1) = 1+b \neq 0$. We can give a convenient rational parametrization of that conic as follows. Since $q \equiv 1 \pmod 3$, (-3) is a quadratic residue in $\mathbb{F}_q$. Thus, $(z_0, w_0) = (\sqrt{-3}/2, 0)$ is an $\mathbb{F}_q$-rational point on the conic (6). We parametrize all the other points by setting $z = z_0 + tw$, which gives:

$$\sqrt{-3} \cdot t + t^2 \cdot w + (1+b) \cdot w = 0,$$

and hence:

$$y = \frac{1}{w} = -\frac{1+b+t^2}{\sqrt{-3} \cdot t}$$
$$v = z_0 + tw - 1 = \frac{-1+\sqrt{-3}}{2} + \frac{\sqrt{-3} \cdot t^2}{1+b+t^2}.$$

This is well-defined (and y is nonzero) if and only if $t \neq 0$ and $t^2 \neq -1-b$, and the second condition is always verified since $\chi_q(-1-b) = -\chi_q(1+b) = -1$. Thus, for any $t \neq 0$, it follows from Lemma 3 that at least one of the three values:

$$x_1 = v = \frac{-1+\sqrt{-3}}{2} - \frac{\sqrt{-3} \cdot t^2}{1+b+t^2}, \tag{7}$$
$$x_2 = -1 - v = \frac{-1-\sqrt{-3}}{2} + \frac{\sqrt{-3} \cdot t^2}{1+b+t^2}, \tag{8}$$
$$x_3 = 1 + y^2 = 1 - \frac{(1+b+t^2)^2}{3t^2} \tag{9}$$

is the abscissa of a point in $E(\mathbb{F}_q)$. Furthermore, we see that these values only depend on t^2, and hence are invariant under a change of sign for t. As a result, it is natural to map t and $-t$ to opposite points on $E(\mathbb{F}_q)$ with one of the previous coordinates.

Therefore, we can define the Shallue–van de Woestijne encoding to the BN curve E as follows.

Definition 2. *For all $t \in \mathbb{F}_q^*$, let $x_1, x_2, x_3 \in \mathbb{F}_q$ be as in Eqs. (7) to (9). The* SW encoding *to the BN curve E is the map:*

$$f\colon \mathbb{F}_q^* \longrightarrow E(\mathbb{F}_q)$$
$$t \longmapsto \left(x_i,\ \chi_q(t) \cdot \sqrt{g(x_i)}\right),$$

where for each t, $i \in \{1,2,3\}$ is the smallest index such that $g(x_i)$ is a square in $\mathbb{F}_q$.

The encoding can be extended to all of $\mathbb{F}_q$ by sending 0 to some arbitrary point in $E(\mathbb{F}_q)$. Since x_1 is well-defined and equal to $(-1+\sqrt{-3})/2$ for $t = 0$, and $g(x_1) = 1+b$ is a square, a relatively natural choice may be to set:

$$f(0) = \left(\frac{-1+\sqrt{-3}}{2},\ \sqrt{1+b}\right).$$

4 Computing the Image Size

In this section, we estimate the number of points in the image of the Shallue–van de Woestijne encoding f to $E(\mathbb{F}_q)$, using a refinement of the techniques from [20,22]. We will show that f reaches roughly 9/16ths of all points on the curve, or more precisely, that $\#f(\mathbb{F}_q^*) = (9/16) \cdot q + O(\sqrt{q})$ (where the constant in the big-O is universal and will be made explicit).

To obtain that estimate, we first write $\mathbb{F}_q^*$ as the disjoint union of the subsets T_1, T_2, T_3 of field elements t such that the corresponding index i in Definition 2 is 1, 2, 3 respectively. In other words:

$$\begin{aligned} T_1 &= \{t \in \mathbb{F}_q^* \mid g(x_1) \text{ is a square}\}; \\ T_2 &= \{t \in \mathbb{F}_q^* \mid g(x_1) \text{ is not a square but } g(x_2) \text{ is}\}; \\ T_3 &= \{t \in \mathbb{F}_q^* \mid \text{neither } g(x_1) \text{ nor } g(x_2) \text{ are squares}\}. \end{aligned}$$

Then, we examine the points in $f(T_i)$ for $i = 1, 2, 3$.

Clearly, a point $(x, y) \in f(T_1)$ satisfies:

$$x = x_1(t) = \frac{-1 + \sqrt{-3}}{2} - \frac{\sqrt{-3} \cdot t^2}{1 + b + t^2}$$

for some $t \neq 0$, or equivalently:

$$t^2 = -\frac{(1+b) \cdot (x - \zeta)}{x - \zeta^2} \quad \text{where} \quad \zeta = \frac{-1 + \sqrt{-3}}{2}. \tag{10}$$

We denote by $\omega_1 \in \mathbb{F}_q(x)$ the rational function on the right-hand side of Eq. (10). The set $f(T_1)$ is thus contained in the set of points $(x, y) \in E(\mathbb{F}_q)$ such that ω_1 is a nonzero square. And conversely, if $(x, y) \in E(\mathbb{F}_q)$ satisfies that $\omega_1 = t^2$ for some $t \neq 0$, we get $x = x_1(t)$ and hence $(x, y) = f(t)$ or $f(-t)$ depending on the sign of $\chi_q(y)$.

Thus, we obtain that $f(T_1)$ is the set of points $(x, y) \in E(\mathbb{F}_q)$ such that ω_1 is a nonzero square. If we denote by K the function field of E, this set is thus in bijection with the set of places of degree 1 in K that split in the quadratic extension L_1/K with $L_1 = K[t]/(t^2 - \omega_1)$. We can thus apply Lemma 1 with $s = 1$ and $m = d = 2$ to get:

$$\left| \#f(T_1) - \frac{1}{2}q \right| \leq (2 + g_{L_1}) \cdot q^{1/2} + 6q^{1/4} + g_{L_1} + 4,$$

where g_{L_1} is the genus of the function field L_1. After extending the field of scalars to $\overline{\mathbb{F}}_q$, we can see that L_1 is ramified above exactly 4 places of K (corresponding to the two opposite points in $E(\overline{\mathbb{F}}_q)$ where the rational function ω_1 vanishes and the two others where it has a pole), so the Riemann–Hurwitz formula gives $g_{L_1} = 3$. Hence:

$$\left| \#f(T_1) - \frac{1}{2}q \right| \leq 5q^{1/2} + 6q^{1/4} + 7. \tag{11}$$

Similarly, a point $(x, y) \in f(T_2)$ satisfies:

$$x = x_2(t) = \frac{-1 - \sqrt{-3}}{2} + \frac{\sqrt{-3} \cdot t^2}{1 + b + t^2},$$

for some $t \neq 0$, or equivalently:

$$t^2 = -\frac{(1+b)(x - \zeta^2)}{x - \zeta} \quad \text{where, again,} \quad \zeta = \frac{-1 + \sqrt{-3}}{2}. \tag{12}$$

Thus, for any point $(x, y) \in f(T_2)$, the rational function ω_2 on the right-hand side of Eq. (12) is a nonzero square. But we clearly have:

$$\omega_1 = \frac{(1+b)^2}{\omega_2},$$

and it follows that if ω_2 is a nonzero square, then so is ω_1. As a result, we must have $f(T_2) \subset f(T_1)$.[4] Therefore:

$$f(\mathbb{F}_q^*) = f(T_1) \cup f(T_2) \cup f(T_3) = f(T_1) \cup \big(f(T_3) \setminus f(T_1)\big)$$

and we can thus complete our estimate if we can evaluate the cardinality of $f(T_3) \setminus f(T_1)$.

Again, a point $(x, y) \in E(\mathbb{F}_q)$ is in $f(T_3)$ if and only if there exists some $t \in \mathbb{F}_q^*$ such that neither $g(x_1(t))$ nor $g(x_2(t))$ is a square, and $x = x_3(t) = 1 - (1 + b + t^2)^2/(3t^2)$. The last relation is equivalent to:

$$t^4 + \big[3(x-1) + 2(1+b)\big]t^2 + (1+b)^2 = 0. \tag{13}$$

As a biquadratic polynomial over the function field K of E, that polynomial in t, which we denote $P(t)$, is clearly irreducible and has Galois group V_4 (since its constant coefficient is a square; see [31, Th. 2 and 3]). In particular, $L_3' = K[t]/(P)$ is a Galois extension of K, and its 4 automorphisms send t to $\pm t$ and $\pm(1+b)/t$.

Now, the set $f(T_3)$ is in bijection with the set of places of K where the polynomial P has at least one root t (necessarily nonzero; and in that case, P splits completely) that further satisfies that neither $g(x_1(t))$ nor $g(x_2(t))$ is a square, or equivalently, that both $-g(x_1(t))$ and $-g(x_2(t))$ are nonzero squares. By construction of the Shallue–van de Woestijne encoding, we know that $g(x_1(t)) \cdot g(x_2(t)) \cdot g(x_3(t))$ is a nonzero square for all $t \in \mathbb{F}_q^*$, so that for any t as above, $-g(x_1(t))$ is a nonzero square if and only if $-g(x_2(t))$ is a nonzero square. Furthermore, the automorphisms of $L_3' = K[t]/(P)$ that send t to $\pm t$ fix x_1 and x_2, whereas those that send t to $\pm(1+b)/t$ exchange them, as we can see from the fact that $\omega_2 = (1+b)^2/\omega_1$. This ensures that the quadratic extension

[4] It does not matter for our purposes, but that inclusion is usually strict: indeed, $f(T_2)$ is smaller that the set of points in $E(\mathbb{F}_q)$ with an abscissa of the form $x_2(t)$, because the corresponding parameter t must satisfy the additional condition that $g(x_1(t))$ is not a square.

$L_3'' = L_3'[z]/\big(z^2 + g(x_1(t))\big)$ of L_3' is in fact Galois of degree 8 over K, and $f(T_3)$ is in bijection with the set of places of K that split completely in L_3''.

Finally, an element $(x, y) \in f(T_3)$ is *not* in $f(T_1)$ exactly when ω_1 isn't a nonzero square, i.e. when $-\omega_1$ is a square. As a result, up to possibly two points where $-\omega_1$ vanishes, $f(T_3) \setminus f(T_1)$ has the same number of elements as the set of places in K which split completely in the compositum $L_3 = L_3'' \cdot K[w]/(w^2 + \omega_1)$ (a Galois extension of degree 16, since the two fields are linearly disjoint by inspection of their ramification, as seen below). Thus, we can apply Lemma 1 with $s = 1$, $d = 2$ and $m = 16$ to get:

$$\left| \#\big(f(T_3) \setminus f(T_1)\big) - \frac{1}{16}q \right| \leq \left(2 + \frac{g_{L_3}}{8}\right) \cdot q^{1/2} + 6q^{1/4} + \frac{g_{L_3}}{8} + 4 + 2,$$

where g_{L_3} is the genus of the function field L_3. To compute that genus, we examine the ramification of the various fields involved, after an extension of scalars to $\overline{\mathbb{F}}_q$. Clearly, $(\overline{\mathbb{F}}_q K)[w]/(w^2 + \omega_1)$ is simply ramified over the four places corresponding to the points in $E(\overline{\mathbb{F}}_q)$ with $x = \zeta$ or ζ^2. Thus, that field has genus 4 again. On the other hand, since the discriminant of P is:

$$\Delta = 48 \cdot (1 + b)^2 \cdot (x - 1)^2 \cdot \big(3(x - 1) + 4(1 + b)\big)^2,$$

the field $\overline{\mathbb{F}}_q L_3'$ is ramified with ramification type $(2, 2)$ over the places corresponding to the points in $E(\overline{\mathbb{F}}_q)$ with $x = 1$ or $x = -(1 + 4b)/3$. In turn, $\overline{\mathbb{F}}_q L_3''$ is ramified over the places in $\overline{\mathbb{F}}_q L_3'$ where $x_1(t)^3 = -b$ or $x_1(t) = \infty$. This gives 8 values of t, or 16 places of $\overline{\mathbb{F}}_q L_3'$ (since each value of t corresponds to one value of x and two of y). Putting everything together and using Abhyankar's lemma, we obtain that the ramification divisor of $\overline{\mathbb{F}}_q L_3$ over $\overline{\mathbb{F}}_q K$ has degree $4 \cdot 8 + (4 \cdot 2 \cdot 2 + 16) \cdot 2 = 96$. Thus, the Riemann–Hurwitz formula gives $2g_{L_3} - 2 = 16 \cdot 0 + 96$, hence $g_{L_3} = 49 < 7 \cdot 8$, and thus:

$$\left| \#\big(f(T_3) \setminus f(T_1)\big) - \frac{1}{16}q \right| \leq 9q^{1/2} + 6q^{1/4} + 13. \tag{14}$$

Combining Eqs. (11) and (14), we get the following result, as expected.

Theorem 1. *The number of points in the image $f(\mathbb{F}_q^*)$ of the Shallue–van de Woestijne encoding to a BN curve is bounded as:*

$$\left| \#f(\mathbb{F}_q^*) - \frac{9}{16}q \right| \leq 14q^{1/2} + 12q^{1/4} + 20.$$

Remark 1. While somewhat arbitrary, the numbering of the points x_1, x_2, x_3 in the definition of the Shallue–van de Woestijne encoding actually matters for the computation of the number of points: for example, it is not difficult to adapt the argument above to see that if the order was reversed, the image size would only be about $7/16 \cdot q$ instead of $9/16 \cdot q$.

5 Obtaining Indifferentiability

In this section, we prove that, while f itself is clearly not an admissible encoding in the sense of Section 2.3, the tensor square $f^{\otimes 2}$, as defined in Section 2.4, is indeed admissible, and hence the hash function:

$$m \mapsto f\big(\mathfrak{h}_1(m)\big) + f\big(\mathfrak{h}_2(m)\big) \tag{15}$$

is indifferentiable from a random oracle when $\mathfrak{h}_1, \mathfrak{h}_2$ are seen as independent random oracles to $\mathbb{F}_q^*$.

To see this, first note that $f^{\otimes 2}$ is obviously efficiently computable, and it is also samplable: a sketch of a sampling function is as follows. To find a uniformly random preimage (u, v) of some point $\mathbf{P} \in E(\mathbb{F}_q)$, pick $v \in \mathbb{F}_q^*$ at random, and find all the preimages of $\mathbf{P} - f(v)$ (which can be done by solving three algebraic equations, corresponding to the three "branches" of f). There are at most 4 such preimages. Then pick $i \in \{1, 2, 3, 4\}$ at random and return the i-th preimage if it exists. Otherwise, start over with another v. The image size computation of the previous section guarantees that the expected number of iterations is finite, which ensures samplability. See [5] for a complete formal treatment of the samplability of Icart's function, which is easily adapted to our case along the lines of the previous sketch.

Thus, all that remains to see to prove admissibility is that $f^{\otimes 2}$ is regular. We will show that using the results of Section 2.4, by proving that f is a *well-distributed* encoding. We have to bound the following sum:

$$S_f(\chi) = \sum_{t \in \mathbb{F}_q^*} \chi(f(t))$$

for every nontrivial character χ of $E(\mathbb{F}_q)$. As in the previous section, we break the sum into sums over T_1, T_2 and T_3 which we treat separately.

To estimate the sum over T_1, we introduce the covering curve $h_1 \colon X_1 \to E$ corresponding to the extension of function fields L_1/K (with the notation of Section 4). In other words, a rational point in $X_1(\mathbb{F}_q)$ is a tuple (x, y, t) such that $(x, y) \in E(\mathbb{F}_q)$ and $x = x_1(t)$ (or equivalently $t^2 = \omega_1(x)$). In particular, for any $t \in T_1$, there are two rational points of X_1 whose third coordinate is t: if we let $(x, y) = f(t)$, these two points are (x, y, t) and $(x, -y, t)$, which map to $\chi(f(t))$ and $\chi(f(t))^{-1}$ under $\chi \circ h_1$. Thus, we get:

$$\sum_{\mathbf{P} \in X_1(\mathbb{F}_q)} \chi(h_1(\mathbf{P})) = \sum_{t \in T_1} \chi(f(t)) + \sum_{t \in T_1} \chi(f(t))^{-1} + O(1),$$

where the constant $O(1)$ accounts for a bounded number of exceptional points (ramification, points at infinity). We would like to get rid of the second sum on the right-hand side. For that purpose, note that the "correct" y value corresponding to a given t is the one such that $\chi_q(ty) = 1$. It follows that:

$$\sum_{\mathbf{P} \in X_1(\mathbb{F}_q)} \frac{1 + \chi_q(ty)}{2} \chi(h_1(\mathbf{P})) = \sum_{t \in T_1} \chi(f(t)) + O(1),$$

and hence, by Lemma 2:

$$\left|\sum_{t\in T_1}\chi(f(t))\right| \leq \big(2g_{X_1} - 2 + \deg_{X_1}(ty)\big)\cdot\sqrt{q} + O(1) = 12\sqrt{q} + O(1), \tag{16}$$

since $g_{X_1} = g_{L_1} = 3$ as seen before, and the rational functions t and y over X_1 are of degree 2 and 6 respectively.

Similarly, to estimate the character sum:

$$\sum_{t\in T_2}\chi(f(t)), \tag{17}$$

we introduce the extension $L_2 = K[t,z]/\big(t^2-\omega_2, z^2+g(x_1(t))\big)$ of K associated to f over T_2, and the corresponding covering curve $h_2\colon X_2 \to E$. A point in $X_2(\mathbb{F}_q)$ is thus a tuple (x,y,t,z) such that $(x,y)\in E(\mathbb{F}_q)$, $x = x_2(t)$, and z is a square root of $-g(x_1(t))$ ensuring that $g(x_1(t))$ is not a square. For a given $t\in T_2$, there are four rational points of X_1 whose third coordinate is t, namely $(x,\pm y,t,\pm z)$ with $(x,y) = f(t)$ and $z = \sqrt{-g(x_1(t))}$. As with T_1, we can thus write the character sum (17) as:

$$\sum_{t\in T_2}\chi(f(t)) = \frac{1}{2}\sum_{\mathbf{P}\in X_2(\mathbb{F}_q)}\frac{1+\chi_q(ty)}{2}\chi(h_2(\mathbf{P})) + O(1),$$

where the factor $1/2$ accounts for the two values of z. By Lemma 2, it follows that:

$$\left|\sum_{t\in T_2}\chi(f(t))\right| \leq \frac{1}{2}\big(2g_{X_2} - 2 + \deg_{X_2}(ty)\big)\cdot\sqrt{q} + O(1) = 20\sqrt{q} + O(1), \tag{18}$$

since $g_{X_2} = g_{L_2} = 13$ by inspection of the ramification, and the rational functions t and y over X_2 are of degree 4 and 12 respectively.

Finally, to estimate the character sum:

$$\sum_{t\in T_3}\chi(f(t)),$$

we introduce the covering curve $h_3\colon X_3 \to E$ corresponding to the extension $L_3'' = K[t,z]/\big(P(t), z^2+g(x_1(t))\big)$ of K defined in Section 4. The expression of the character sum is the same as with T_2:

$$\sum_{t\in T_3}\chi(f(t)) = \frac{1}{2}\sum_{\mathbf{P}\in X_3(\mathbb{F}_q)}\frac{1+\chi_q(ty)}{2}\chi(h_3(\mathbf{P})) + O(1).$$

By Lemma 2, it follows that:

$$\left|\sum_{t\in T_3}\chi(f(t))\right| \leq \frac{1}{2}\big(2g_{X_3} - 2 + \deg_{X_3}(ty)\big)\cdot\sqrt{q} + O(1) = 30\sqrt{q} + O(1), \tag{19}$$

since $g_{X_3} = g_{L_3''} = 17$ by inspection of the ramification, and the rational functions t and y over X_2 are of degree 4 and 24 respectively.

Putting Eqs. (16), (18) and (19) together, we obtain that $|S_f(\chi)| \leq 62\sqrt{q} + O(1)$ for any nontrivial character χ, and hence f is well-distributed as required.

Using the statistical distance bound given in Section 2.4 together with [13, Th. 1], it follows that for a $2k$-bit BN curve, the hash function given by Eq. (15) is ε-indifferentiable from a random oracle, where:

$$\varepsilon = 4 \cdot \big(62 + O(q^{-1/2})\big)^2 \cdot \frac{\sqrt{\#E(\mathbb{F}_q)}}{q} \cdot q_D \leq \big(2^{14} + o(1)\big)2^{-k} \cdot q_D$$

if we denote by q_D the number of queries made by the distinguisher.

6 Efficient Computation

Finally, we would like to describe a possible implementation of the Shallue–van de Woestijne encoding from Definition 2 that is both efficient and secure against side-channel analysis and other physical attacks. It is not difficult to meet what is more or less the standard of efficiency for elliptic curve encodings, as set by functions like Icart's [26]—namely, that an evaluation of the function should cost one exponentiation in the base field, plus a small, bounded number of faster operations. In the case of our encoding f, we simply need to compute the values x_1, x_2, x_3, and decide, based on $\chi_q(g(x_1))\,, \chi_q(g(x_2))$, which of those three values will be the abscissa of the output point.

This simple implementation has two problems with respect to side-channel attacks, however.

One the one hand, computing the quadratic character is difficult to do in constant time, so the length of that part of the computation may leak information about the input. We propose to alleviate that problem using blinding: instead of computing $\chi_q(g(x_1))$, we evaluate $\chi_q\big(r_1^2 \cdot g(x_1)\big)$ for some random $r_1 \in \mathbb{F}_q^*$. If we then make sure that the quadratic character is implemented in such a way that evaluating $\chi_q(a)$ and $\chi_q(-a)$ takes the same time (which isn't hard to achieve), the duration of the computation of the two quadratic characters we need is completely independent of the input.

On the other hand, the naive way to choose the index i of the output abscissa involves several conditional branches. This opens up a (small) risk of timing attacks, as well as (more serious) possibilities for fault injection (i.e. glitch attacks). We avoid that problem by selecting the index using an algebraic formula depending on the two quadratic character values. It suffices to construct a function ψ of two variables such that:

$$\psi(1,1) = \psi(1,-1) = 1, \quad \psi(-1,1) = 2, \quad \psi(-1,-1) = 3.$$

One such function is given by:

$$\psi(\alpha,\beta) = \big[(\alpha - 1) \cdot \beta \bmod 3\big] + 1.$$

Using that function, we propose the implementation of the encoding given in Algorithm 1.

Algorithm 1. Shallue–van de Woestijne encoding to BN curves.

procedure SWEncBN(t) ▷ $t \in \mathbb{F}_q^*$
  $w \leftarrow \sqrt{-3} \cdot t/(1 + b + t^2)$
  $x_1 \leftarrow (-1 + \sqrt{-3})/2 - tw$
  $x_2 \leftarrow -1 - x_1$
  $x_3 \leftarrow 1 + 1/w^2$
  $r_1, r_2, r_3 \xleftarrow{\$} \mathbb{F}_q^*$
  $\alpha \leftarrow \chi_q\big(r_1^2 \cdot (x_1^3 + b)\big)$
  $\beta \leftarrow \chi_q\big(r_2^2 \cdot (x_2^3 + b)\big)$
  $i \leftarrow \big[(\alpha - 1) \cdot \beta \bmod 3\big] + 1$
  return $\big(x_i, \chi_q\big(r_3^2 \cdot t\big) \cdot \sqrt{x_i^3 + b}\big)$
end procedure

Acknowledgments. We would like to thank Paulo Barreto for suggesting this problem, Sorina Ionica and himself for fruitful discussions, and the reviewers of LATINCRYPT 2012 for numerous useful comments.

References

1. Baek, J., Zheng, Y.: Identity-based threshold decryption. In: Bao et al. [2], pp. 262–276
2. Bao, F., Deng, R., Zhou, J. (eds.): PKC 2004. LNCS, vol. 2947. Springer, Heidelberg (2004)
3. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing Elliptic Curves with Prescribed Embedding Degrees. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 257–267. Springer, Heidelberg (2003)
4. Barreto, P.S.L.M., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
5. Barthe, G., Grégoire, B., Heraud, S., Olmedo, F., Zanella Béguelin, S.: Verified Indifferentiable Hashing into Elliptic Curves. In: Degano, P., Guttman, J.D. (eds.) Principles of Security and Trust. LNCS, vol. 7215, pp. 209–228. Springer, Heidelberg (2012)
6. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-group signature scheme. In: Desmedt [17], pp. 31–46
7. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
8. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
9. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
10. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. J. Cryptology 17(4), 297–319 (2004)

11. Boyen, X.: Multipurpose Identity-Based Signcryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 383–399. Springer, Heidelberg (2003)
12. Boyko, V., MacKenzie, P.D., Patel, S.: Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000)
13. Brier, E., Coron, J.-S., Icart, T., Madore, D., Randriam, H., Tibouchi, M.: Efficient Indifferentiable Hashing into Ordinary Elliptic Curves. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 237–254. Springer, Heidelberg (2010)
14. Cha, J.C., Cheon, J.H.: An identity-based signature from Gap Diffie-Hellman groups. In: Desmedt [17], pp. 18–30
15. Chevallier-Mames, B.: An Efficient CDH-Based Signature Scheme with a Tight Security Reduction. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 511–526. Springer, Heidelberg (2005)
16. Couveignes, J.-M., Kammerer, J.-G.: The geometry of flex tangents to a cubic curve and its parameterizations. Journal of Symbolic Computation 47(3), 266–281 (2012)
17. Desmedt, Y.G. (ed.): PKC 2003. LNCS, vol. 2567. Springer, Heidelberg (2002)
18. Farashahi, R.R.: Hashing into Hessian Curves. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 278–289. Springer, Heidelberg (2011)
19. Farashahi, R.R., Fouque, P.-A., Shparlinski, I.E., Tibouchi, M., Voloch, J.F.: Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. Math. Comput. (to appear, 2012)
20. Farashahi, R.R., Shparlinski, I.E., Voloch, J.F.: On hashing into elliptic curves. J. Math. Cryptology 3, 353–360 (2010)
21. Fouque, P.-A., Tibouchi, M.: Deterministic encoding and hashing to odd hyperelliptic curves. In: Joye et al. [28], pp. 265–277
22. Fouque, P.-A., Tibouchi, M.: Estimating the Size of the Image of Deterministic Hash Functions to Elliptic Curves. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 81–91. Springer, Heidelberg (2010)
23. Fried, M.D., Jarden, M.: Field arithmetic, 2nd edn. Ergebnisse der Mathematik und ihrer Grenzgebiete, vol. 11. Springer, Berlin (2005)
24. Gentry, C., Silverberg, A.: Hierarchical ID-based cryptography. In: Zheng [43], pp. 548–566
25. Horwitz, J., Lynn, B.: Toward Hierarchical Identity-Based Encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 466–481. Springer, Heidelberg (2002)
26. Icart, T.: How to Hash into Elliptic Curves. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 303–316. Springer, Heidelberg (2009)
27. Jablon, D.P.: Strong password-only authenticated key exchange. SIGCOMM Comput. Commun. Rev. 26, 5–26 (1996)
28. Groth, J.: Pairing-Based Non-interactive Zero-Knowledge Proofs. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 206–206. Springer, Heidelberg (2010)
29. Kachisa, E.J., Schaefer, E.F., Scott, M.: Constructing Brezing-Weng Pairing-Friendly Elliptic Curves Using Elements in the Cyclotomic Field. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209, pp. 126–135. Springer, Heidelberg (2008)
30. Kammerer, J.-G., Lercier, R., Renault, G.: Encoding points on hyperelliptic curves over finite fields in deterministic polynomial time. In: Joye et al. [28], pp. 278–297
31. Kappe, L.-C., Warren, B.: An elementary test for the Galois group of a quartic polynomial. Amer. Math. Monthly 96(2), 133–137 (1989)

32. Libert, B., Quisquater, J.-J.: Efficient signcryption with key privacy from Gap Diffie-Hellman groups. In: Bao et al. [2], pp. 187–200
33. 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)
34. Pereira, G.C.C.F., Simplício Jr., M.A., Naehrig, M., Barreto, P.S.L.M.: A family of implementation-friendly BN elliptic curves. The Journal of Systems and Software 84(8), 1319–1326 (2011)
35. 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)
36. Sato, H., Hakuta, K.: An efficient method of generating rational points on elliptic curves. J. Math-for-Industry 1(A), 33–44 (2009)
37. Schinzel, A., Skałba, M.: On equations $y^2 = x^n + k$ in a finite field. Bull. Pol. Acad. Sci. Math. 52(3), 223–226 (2004)
38. Shallue, A., van de Woestijne, C.E.: Construction of Rational Points on Elliptic Curves over Finite Fields. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 510–524. Springer, Heidelberg (2006)
39. Skałba, M.: Points on elliptic curves over finite fields. Acta Arith. 117, 293–301 (2005)
40. Tibouchi, M.: Hachage vers les courbes elliptiques et cryptanalyse de schémas RSA. PhD thesis, Univ. Paris 7 and Univ. Luxembourg, Introduction in French, main matter in English (2011)
41. Tibouchi, M.: A note on hasing to BN curves. In: Miyaji, A. (ed.) SCIS. IEICE (2012)
42. Zhang, F., Kim, K.: ID-based blind signature and ring signature from pairings. In: Zheng [43], pp. 533–547
43. Zheng, Y. (ed.): ASIACRYPT 2002. LNCS, vol. 2501. Springer, Heidelberg (2002)

Semi-bent Functions with Multiple Trace Terms and Hyperelliptic Curves

Sihem Mesnager

LAGA (Laboratoire Analyse, Géometrie et Applications), UMR 7539, CNRS,
Department of Mathematics,University of Paris XIII and University of Paris VIII,
2 rue de la liberté, 93526 Saint-Denis Cedex, France
smesnager@univ-paris8.fr

Abstract. Semi-bent functions with even number of variables are a class of important Boolean functions whose Hadamard transform takes three values. Semi-bent functions have been extensively studied due to their applications in cryptography and coding theory. In this paper we are interested in the property of semi-bentness of Boolean functions defined on the Galois field $\mathbb{F}_{2^n}$ (n even) with multiple trace terms obtained via Niho functions and two Dillon-like functions (the first one has been studied by the author and the second one has been studied very recently by Wang et al. using an approach introduced by the author). We subsequently give a connection between the property of semi-bentness and the number of rational points on some associated hyperelliptic curves. We use the hyperelliptic curve formalism to reduce the computational complexity in order to provide an efficient test of semi-bentness leading to substantial practical gain thanks to the current implementation of point counting over hyperelliptic curves.

Keywords: Boolean function, Symmetric cryptography, Walsh-Hadamard transformation, Semi-bent functions, Dickson polynomial, Hyperelliptic curves.

1 Introduction

A number of research works in symmetric cryptography are devoted to problems of resistance of various ciphering algorithms to the fast correlation attacks (on stream ciphers) and to the linear cryptanalysis (on block ciphers). These works analyse various classes of approximating functions and constructions of functions with the best resistance to such approximations. Some general classes of Boolean functions play a central role with this respect: the class of *bent functions* [16], that is, of Boolean functions of an even number of variables that have the maximum possible Hamming distance to the set of all affine functions, its subclasses of homogeneous bent functions and hyper-bent functions and the generalizations of the notion: semi-bent functions [4], Z-bent functions, negabent functions, etc.

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 18–36, 2012.

The paper is devoted to *semi-bent* Boolean functions. The notion of semi-bent function has been introduced by Chee, Lee and Kim [4] at Asiacrypt' 94. These functions had been previously investigated under the name of three-valued almost optimal Boolean functions in [1]. Moreover, they are particular cases of the so-called plateaued functions [20]. Semi-bent functions are widely studied in cryptography because, besides having low Hadamard transform which provides protection against fast correlation attacks [12] and linear cryptanalysis [11], they can possess desirable properties such as low autocorrelation, propagation criteria, resiliency and high algebraic degree. Semi-bent functions have been paid a lot of attention in code division multiple access (CDMA) communication systems for sequence design. Semi-bent functions exist for even or odd number of variables. When n is even, the semi-bent functions are those Boolean functions whose Hadamard transform takes values 0 and $\pm 2^{\frac{n+2}{2}}$. They are balanced (up to the addition of a linear function) and have maximal non-linearity among balanced plateaued functions. Results concerning quadratic semi-bent functions with even number of inputs can be found in [3]. Links of semi-bent functions from Dillon and Niho exponents with exponential sums (namely, Kloosterman sums) can be found in [15]. Some constructions of monomial (that is, absolute trace of a power function) semi-bent functions (namely, quadratic functions) have been proposed in [17]. Recently, a large number of infinite classes of semi-bent functions in explicit bivariate (resp. univariate) polynomial form have been obtained in [2].

In this paper, functions in univariate representation expressed by means of trace functions via Dillon-like exponents (proposed by the author [14] and very recently by Wang et al. [19]) and Niho exponents with even number of variables are considered. Our main intention is to provide an efficient characterization of the semi-bentness property of the corresponding functions (whose expressions are in polynomial forms with multiple trace terms). To this end, we precise firstly the connection between the semi-bentness property of such functions and some exponential sums involving Dickson polynomials. Next, in the line of the recent works of Lisonek [10] and further of Flori and Mesnager [8], we give a link between the property of semi-bentness and the number of rational points on certain hyperelliptic curves. The paper exploits the connections between semi-bentness property and binary hyperelliptic curves to produce a polynomial complexity test which is of use in constructing semi-bent functions with multiple trace terms.

The paper is organized as follows. In section 2, we fix our main notation and recall the necessary background. In section 3, we investigate the link between the semi-bentness property of some infinite classes of Boolean functions in univariate representation and some exponential sums involving Dickson polynomials. Such a link leads to an exponential time test of semi-bentness. Finally, in section 4, we connect the property of semi-bentness of such functions to hyperelliptic curves and we reformulate the characterization obtained in section 3 in terms of cardinalities of hyperelliptic curves leading to an efficient test of semi-bentness and practical speed-ups.

2 Notation and Preliminaries

2.1 Boolean Functions in Polynomial Forms

Let n be a positive integer. A Boolean function f on $\mathbb{F}_{2^n}$ is an $\mathbb{F}_2$-valued function on the Galois field $\mathbb{F}_{2^n}$ of order 2^n. The *weight* of f, denoted by $wt(f)$, is the *Hamming weight* of the image vector of f, that is, the cardinality of its support $\{x \in \mathbb{F}_{2^n} \mid f(x) = 1\}$.

For any positive integer k, and for any r dividing k, the trace function from $\mathbb{F}_{2^k}$ to $\mathbb{F}_{2^r}$, denoted by Tr_r^k, is the mapping defined as:

$$\forall x \in \mathbb{F}_{2^k}, \quad Tr_r^k(x) := \sum_{i=0}^{\frac{k}{r}-1} x^{2^{ir}} = x + x^{2^r} + x^{2^{2r}} + \cdots + x^{2^{k-r}}.$$

In particular, the *absolute trace* over $\mathbb{F}_2$ is the function $Tr_1^n(x) = \sum_{i=0}^{n-1} x^{2^i}$. Recall that, for every integer r dividing k, the trace function Tr_r^k satisfies the transitivity property, that is, $Tr_1^k = Tr_1^r \circ Tr_r^k$.

There exist several kinds of possible trace (univariate) representations of Boolean functions which are not necessary unique and use the identification between the vector-space $\mathbb{F}_2^n$ and the field $\mathbb{F}_{2^n}$.

Every non-zero Boolean function f defined on $\mathbb{F}_{2^n}$ has a (unique) trace expansion of the form:

$$\forall x \in \mathbb{F}_{2^n}, \quad f(x) = \sum_{j \in \Gamma_n} Tr_1^{o(j)}(a_j x^j) + \epsilon(1 + x^{2^n-1})$$

where Γ_n is the set of integers obtained by choosing one element in each cyclotomic coset of 2 modulo $2^n - 1$ (the most usual choice for j is the smallest element in its cyclotomic class, called the coset leader of the class), $o(j)$ is the size of the cyclotomic coset of 2 modulo $2^n - 1$ containing j, $a_j \in \mathbb{F}_{2^{o(j)}}$ and $\epsilon = wt(f)$ modulo 2.

This trace representation of f is unique and is called its *polynomial form.*

2.2 Walsh Transform and Semi-bent Functions

Let f be a Boolean function on $\mathbb{F}_{2^n}$. Its *"sign" function* is the integer-valued function $\chi_f := (-1)^f$. The *W*alsh Hadamard transform of f is the discrete Fourier transform of χ_f, whose value at $\omega \in \mathbb{F}_{2^n}$ is defined as:

$$\forall \omega \in \mathbb{F}_{2^n}, \quad \widehat{\chi_f}(\omega) = \sum_{x \in \mathbb{F}_{2^n}} (-1)^{f(x)+Tr_1^n(\omega x)}.$$

Bent functions [16] can be defined as:

Definition 1. *A Boolean function $f : \mathbb{F}_{2^n} \to \mathbb{F}_2$ (n even) is said to be bent if $\widehat{\chi_f}(\omega) = \pm 2^{\frac{n}{2}}$, for all $\omega \in \mathbb{F}_{2^n}$.*

Semi-bent functions [4], [5] are defined as:

Definition 2. *For even n, a Boolean function $f : \mathbb{F}_{2^n} \to \mathbb{F}_2$ is said to be semi-bent if $\widehat{\chi_f}(\omega) \in \{0, \pm 2^{\frac{n+2}{2}}\}$, for all $\omega \in \mathbb{F}_{2^n}$. For odd n, a Boolean function $f : \mathbb{F}_{2^n} \to \mathbb{F}_2$ is said to be semi-bent if $\widehat{\chi_f}(\omega) \in \{0, \pm 2^{\frac{n+1}{2}}\}$, for all $\omega \in \mathbb{F}_{2^n}$.*

2.3 Dickson Polynomial

Recall that the family of binary Dickson polynomials $D_r(X) \in \mathbb{F}_2[X]$ of degree r is defined by $D_r(X) = \sum_{i=0}^{\lfloor \frac{r}{2} \rfloor} \frac{r}{r-i} \binom{r-i}{i} X^{r-2i}, \quad r = 2, 3, \ldots$ Moreover, the family of Dickson polynomials $D_r(X)$ (of degree r) can also be defined by the following recurrence relation: $D_{i+2}(X) = XD_{i+1}(X) + D_i(X)$ with initial values $D_0(X) = 0, \quad D_1(X) = X$. We give the list of the first six Dickson polynomials: $D_0(X) = 0$; $D_1(X) = X$; $D_2(X) = X^2$; $D_3(X) = X + X^3$; $D_4(X) = X^4$; $D_5(X) = X + X^3 + X^5$.

From now, $n = 2m$ is an (even) integer. We denote by U the cyclic group of $(2^m + 1)$-st roots of unity that is $\{u \in \mathbb{F}_{2^n} \mid u^{2^m+1} = 1\}$.

Let R be a subset of representatives of the cyclotomic classes modulo $2^n - 1$ for which each class has the full size n. The author has studied the class of functions whose polynomial form is given by $\sum_{r \in R} Tr_1^n(a_r x^{r(2^m-1)}) + Tr_1^2(bx^{\frac{2^n-1}{3}})$ (where $a_r \in \mathbb{F}_{2^m}^\star$ for $r \in R$ and $b \in \mathbb{F}_4^\star$) and proved a necessary and sufficient condition on the bentness [15] by means of exponential sums involving Dickson polynomials of degrees r and 3 in the case when m is odd. Very recently, Wang et al. have studied (with some restriction) the bentness property of the family of functions whose polynomial form is given by $\sum_{r \in R} Tr_1^n(a_r x^{r(2^m-1)}) + Tr_1^4(b' x^{\frac{2^n-1}{5}})$ (where $a_r \in \mathbb{F}_{2^m}^\star$ for $r \in R$ and $b' \in \mathbb{F}_{16}^\star$). Adopting the approach introduced by the author in [13], Wang et al. proved a necessary and sufficient condition on the bentness [19] by means of exponential sums involving Dickson polynomials of degrees r and 5 in the case when $m \equiv 2 \pmod 4$.

3 Characterizations of Semi-bent Functions with Multiple Trace Terms by Means of Exponential Sums Involving Dickson Polynomials

Nonlinear Boolean functions whose restriction to any vector space $u\mathbb{F}_{2^m}$ (where $u \in U$) are linear are sums of Niho power functions, that is of functions of the form (see [6]):

$$Tr_1^{o((2^m-1)s+1)}\left(a_s x^{(2^m-1)s+1}\right) \text{ with } 1 \leq s \leq 2^m$$

We can determine the value of $o((2^m-1)s+1))$ precisely (recall that $o(j)$ denotes the size of the cyclotomic coset of 2 modulo $2^n - 1$ containing j):

Lemma 1. *([2]) We have $o((2^m - 1)s + 1)) = m$ if $s = 2^{m-1} + 1$ (i.e. if $(2^m - 1)s + 1$ and $2^m + 1$ are conjugate) and $o((2^m - 1)s + 1)) = n$ otherwise.*

Now, consider four infinite classes of functions with multiple trace terms defined on $\mathbb{F}_{2^n}$. We denote by E the set of representatives of the cyclotomic classes modulo 2^n-1 for which each class has full size n. Let $f_{a_r,b,c}$ $f'_{a_r,b}$, $\tilde{f}_{a_r,b',c}$ and $\tilde{f}'_{a_r,b'}$ be the functions defined on $\mathbb{F}_{2^n}$ whose polynomial form is given by (1), (2), (3) and (4), respectively.

$$f_{a_r,b,c}(x) := \sum_{r\in R} Tr_1^n(a_r x^{r(2^m-1)}) + Tr_1^2(bx^{\frac{2^n-1}{3}}) + Tr_1^m(cx^{2^m+1}) \qquad (1)$$

$$f'_{a_r,b}(x) := \sum_{r\in R} Tr_1^n(a_r x^{r(2^m-1)}) + Tr_1^2(bx^{\frac{2^n-1}{3}}) + Tr_1^m(x^{2^m+1}) + Tr_1^n\left(x^{(2^m-1)s+1}\right) \qquad (2)$$

$$\tilde{f}_{a_r,b',c}(x) := \sum_{r\in R} Tr_1^n(a_r x^{r(2^m-1)}) + Tr_1^4(b'x^{\frac{2^n-1}{5}}) + Tr_1^m(cx^{2^m+1}) \qquad (3)$$

$$\tilde{f}'_{a_r,b'}(x) := \sum_{r\in R} Tr_1^n(a_r x^{r(2^m-1)}) + Tr_1^4(b'x^{\frac{2^n-1}{5}}) + Tr_1^m(x^{2^m+1}) + Tr_1^n\left(x^{(2^m-1)s'+1}\right) \qquad (4)$$

where $R \subseteq E$, $a_r \in \mathbb{F}_{2^m}^\star$, $b \in \mathbb{F}_4^\star$, $b' \in \mathbb{F}_{16}^\star$, $c \in \mathbb{F}_{2^m}^\star$, $s \in \{1/4, 3\}$ and $s' \in \{1/6, 3\}$(the fractions $1/4$ and $1/6$ are understood modulo 2^m+1).

Note that $o(r(2^m-1)) = n$, $o(\frac{2^n-1}{3}) = 2$, $o(\frac{2^n-1}{5}) = 4$, $o(2^m+1) = m$ and $o((2^m-1)s+1) = n$ for $s \in \{1/4, 1/6, 3\}$. Moreover, note that for a fixed r coprime with 2^m+1, the function $x \mapsto Tr_1^n(a_r x^{r(2^m-1)})$ is a Dillon function, the function $x \mapsto Tr_1^2(bx^{\frac{2^n-1}{3}})$ (resp. $x \mapsto Tr_1^4(b'x^{\frac{2^n-1}{5}})$) is a Dillon-like function[1] for m odd (resp. for $m \equiv 2 \pmod 4$). Finally, note that for m odd (resp. m even) the function $x \mapsto Tr_1^m(cx^{2^m+1}) + Tr_1^n\left(x^{(2^m-1)s+1}\right)$ (resp. $x \mapsto Tr_1^m(cx^{2^m+1}) + Tr_1^n\left(x^{(2^m-1)s'+1}\right)$) is a Niho bent function [6].

Theorem 1 in [2] ensures that the set of functions defined above by (1), (2), (3) and (4) contains semi-bent functions. In fact, we are interested in these classes because these are the only classes that can be built via known like-Dillon exponents and having the property that they contain semi-bent functions. The goal of this paper is to provide an efficient characterization of the semi-bentness property of the functions $f_{a_r,b,c}$ $f'_{a_r,b}$, $\tilde{f}_{a_r,b',c}$ and $\tilde{f}'_{a_r,b'}$. The first step is to precise a necessary and sufficient condition on the coefficients for a function of the previous form to be semi-bent. In the following, we exhibit a criterion of semi-bentness in terms of exponential sums involving Dickson polynomials for functions in the form (1) and (2). Recall the very recent result on semi-bentness (which is a direct consequence of Theorem 1 in [2]).

[1] We shall call Dillon-like exponents the ones of the form $(2^m-1)s$ where s divides (2^m+1).

Corollary 1. *([2]) Let f be a Boolean function of the form:*

$$f(x) = Tr_1^m(a_0 x^{2^m+1}) + \sum_{i=1}^{L} Tr_1^n(a_i x^{(2^m-1)s_i+1}) + \sum_{r\in R} Tr_1^{o((2^m-1)r)}(b_r x^{(2^m-1)r})$$

where L is some non-negative integer, $2 \le s_i \le 2^m$, $s_i \neq 2^{m-1}+1$, $1 \le r \le 2^m$, $a_0 \in \mathbb{F}_{2^m}$, $a_i \in \mathbb{F}_{2^m}$ and $b_r \in \mathbb{F}_{2^{o((2^m-1)r)}}$ (with at least one coefficient $a_i \neq 0$ and one coefficient $b_r \neq 0$). Assume that:
1) the number of roots u in $U := \{x \in \mathbb{F}_{2^n}; x^{2^m+1} = 1\}$ of the equation $Tr_m^n(cu) + \sum_{i=1}^{L} Tr_m^n(a_i u^{2s_i-1}) + a_0^{\frac{1}{2}} = 0$ is either 0 or 2 for every $c \in \mathbb{F}_{2^n}$,
2) the sum $\sum_{u\in U} \chi(\sum_{r\in R} Tr_1^{o((2^m-1)r)}(b_r u^r))$ is equal to 1.
Then, f is semi-bent.

A proof of the semi-bentness of the function $f_{a_r,b,c}$ is given in [15] by computing the Walsh transform of $f_{a_r,b,c}$. We propose an alternative shorter proof of the following result using Corollary 1 (with $Tr_1^m(x^{-1}) = 0$ for $x = 0$ since $Tr_1^m(x^{-1}) = Tr_1^n(x^{2^{m-1}-1})$).

Proposition 1. *Let $n = 2m$ with m odd. Let $b \in \mathbb{F}_4^\star$, β be a primitive element of $\mathbb{F}_4$ and $c \in \mathbb{F}_{2^m}^\star$. Let $f_{a_r,b,c}$ (resp. $f'_{a_r,b}$) be the function defined on $\mathbb{F}_{2^n}$ whose expression is of the form (1) (resp. form (2)). Let h_{a_r} be the related function defined on $\mathbb{F}_{2^m}$ by $h_{a_r}(x) = \sum_{r\in R} Tr_1^m(a_r D_r(x))$, where $D_r(x)$ is the Dickson polynomial of degree r. Then*

1. *$f_{a_r,\beta,c}$ (resp. $f'_{a_r,\beta}$) is semi-bent if and only if, $f_{a_r,\beta^2,c}$ (resp. f'_{a_r,β^2}) is semi-bent, if and only if,*

$$\sum_{x\in\mathbb{F}_{2^m}} \chi\Big(Tr_1^m(x^{-1}) + h_{a_r}(D_3(x))\Big) = 2^m - 2wt(h_{a_r} \circ D_3) + 4.$$

2. *$f_{a_r,1,c}$ (resp. $f'_{a_r,1}$) is semi-bent if and only if,*

$$3\sum_{x\in\mathbb{F}_{2^m}} \chi\Big(Tr_1^m(x^{-1}) + h_{a_r}(x)\Big) - 2\sum_{x\in\mathbb{F}_{2^m}} \chi\Big(Tr_1^m(x^{-1}) + h_{a_r}(D_3(x))\Big)$$
$$= 4 + 2^m + 4wt(h_{a_r} \circ D_3) - 6wt(h_{a_r}).$$

Proof. Let us study the semi-bentness of $f_{a_r,b,c}$ for $b \in \{1, \beta, \beta^2\}$. Thanks to Corollary 3 in [15], the condition 2) of Corollary 1 (that is, in this case : $\sum_{u\in U} \chi\Big(g_{a_r,b}(u)\Big) = 1$ where $g_{a_r,b} := \sum_{r\in R} Tr_1^n(a_r x^{r(2^m-1)}) + Tr_1^2(bx^{\frac{2^n-1}{3}})$) is satisfied if and only if Condition 1) (resp. Condition 2)) of the proposition is satisfied for $f_{a_r,\beta,c}$ and $f_{a_r,\beta^2,c}$ (resp. $f_{a_r,1,c}$) is satisfied.

On the other hand, we have

$$Tr_m^n(\omega u) + c^{\frac{1}{2}} = 0 \iff \omega u + (\omega u)^{2^m} = c^{\frac{1}{2}} \tag{5}$$

If $\omega = 0$ the equation (5) has no solution. Using the (unique) polar decomposition of $\omega \in \mathbb{F}_{2^n}^\star$: $\omega = vz$, where $v \in U$ and $z \in \mathbb{F}_{2^m}^\star$.

$$\begin{aligned} Tr_m^n(\omega u) + c^{\frac{1}{2}} = 0 &\iff vzu + v^{2^m} z^{2^m} u^{2^m} = c^{\frac{1}{2}} \\ &\iff vu + (uv)^{-1} = z^{-1} c^{\frac{1}{2}}. \end{aligned}$$

Set $c' = cz^{-2}$. Since $u \in U \mapsto vu$ is a permutation on U

$$\#\{u \in U | \omega u + (\omega u)^{2^m} = c^{\frac{1}{2}}\} = \#\{u' \in U | u' + (u')^{2^m} = c'^{\frac{1}{2}}\}$$

Now, if u_1 and u_2 are two distinct solutions in U of the equation $u' + (u')^{2^m} = c'^{\frac{1}{2}}$, then $u_2 = {u_1}^{2^m}$. Indeed, we have $u_1 + {u_1}^{2^m} = c'^{\frac{1}{2}} = u_2 + {u_2}^{2^m}$. Thus, $(u_1 + u_2)^{2^m - 1} = 1$, that is, $u_1 + u_2 \in \mathbb{F}_{2^m}$. Set $u_1 + u_2 = t \in \mathbb{F}_{2^m}$. We have (since $t \in \mathbb{F}_{2^m}$ and $u_1 \in U$): $u_2^{2^m+1} = (t + u_1)^{2^m+1} = (t + u_1)(t + u_1)^{2^m} = (t + u_1)(t^{2^m} + {u_1}^{2^m}) = t^2 + u_1^{2^m} t + u_1 t + 1$.

Now, since $u_2 \in U$, we obtain, $t^2 + u_1^{2^m} t + u_1 t = 0$ that is, $t = 0$ or $t = u_1^{2^m} + u_1$, equivalently, $u_2 = u_1$ or $u_2 = u_1^{2^m}$. Therefore, the number of solutions u in $U := \{u \in \mathbb{F}_{2^n} \mid u^{2^m+1} = 1\}$ of the equation $Tr_m^n(\omega u) + c^{\frac{1}{2}} = 0$ is eiher 0 or 2 for every $\omega \in \mathbb{F}_{2^n}$. Consequently, the Condtion 1) given by Corollary 1 is satisfied.

Now, let us study the semi-bentness of $f'_{a_r,b}$ for $b \in \{1, \beta, \beta^2\}$. Likewise for $f_{a_r,b,c}$, Condition 2) of Corollary 1 holds if and only if Condition 1) (resp. Condition 2)) of the proposition holds for $f'_{a_r,\beta}$ and f'_{a_r,β^2} (resp. $f'_{a_r,1}$). Now, for every $w \in \mathbb{F}_{2^n}$, the equation $Tr_m^n(wu) + Tr_m^n(u^{2s-1}) + 1 = 0$ admits 0 or 2 solutions in U for $s = 1/4$ (with m odd) and for $s = 3$. Indeed, note first that: $Tr_m^n(u^{2s-1}) = Tr_m^n((u^{2s-1})^{2^m}) = Tr_m^n((u^{2^m})^{2s-1}) = Tr_m^n(u^{-1})^{2s-1}) = Tr_m^n(u^{1-2s})$. For $s = 1/4$, the equation $Tr_m^n(wu) + Tr_m^n(u^{1/2}) = 1$ admits 0 or 2 solutions in U for every $w \in \mathbb{F}_{2^n}$ (if m odd) (according to Dobbertin et al. [6]). For $s = 3$, the equation $Tr_m^n(wu) + Tr_m^n(u^5) = 1$ admits 0 or 2 solutions in U for every $w \in \mathbb{F}_{2^n}$ (according to Dobbertin et al. [6]). The proof is complete according to Corollary 1.

Remark 1. Note that one can prove that for every positive integer d such that d is co-prime with $\frac{2^m+1}{3}$, the function defined on $\mathbb{F}_{2^n}$ by $x \mapsto \sum_{r \in R} Tr_1^n(a_r x^{d(2^m-1)}) + Tr_1^2(\beta x^{\frac{2^n-1}{3}}) + Tr_1^m(cx^{2^m+1})$ is also semi-bent if and only if Condition 1) of Proposition 1 holds.

In the following, we exhibit thanks to [19] a criterion of semi-bentness in terms of exponential sums involving Dickson polynomials for functions in the form (3) and (4).

Proposition 2. *Suppose $m := \frac{n}{2} \equiv 2 \pmod 4$. Let $R \subseteq E$ where E is a set of representatives of the cyclotomic classes modulo $2^n - 1$ for which each class has the full size n. Let $b' \in \mathbb{F}_{16}^\star$ and $a_r \in \mathbb{F}_{2^m}^\star$. Let $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) be the function defined on $\mathbb{F}_{2^n}$ whose expression is of the form (3) (resp. form (4)). Let h_{a_r} be the related function defined on $\mathbb{F}_{2^m}$ by $h_{a_r}(x) = \sum_{r \in R} Tr_1^m(a_r D_r(x))$, where $D_r(x)$ is the Dickson polynomial of degree r. Then,*

1. *If b' a primitive element of $\mathbb{F}_{16}$ such that $Tr_1^4(b') = 0$ then, $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if,*

$$\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(D_5(x))\Big) = 2.$$

2. *If $b' = 1$ then, $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if*

$$2\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(D_5(x))\Big) - 5\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(x)\Big) = 4.$$

3. *Assume $a_r \in \mathbb{F}_{2^{\frac{m}{2}}}$. If $b' \in \{\beta, \beta^2, \beta^3, \beta^4\}$ where β is a primitive 5-th root of unity in $\mathbb{F}_{16}$ then, $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if,*

$$\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(D_5(x))\Big) + 5\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(x)\Big) = -8.$$

4. *Assume $a_r \in \mathbb{F}_{2^{\frac{m}{2}}}$. If b' is a primitive element of $\mathbb{F}_{16}$ such that $Tr_1^4(b') = 1$ then, $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if,*

$$3\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(D_5(x))\Big) - 5\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(x)\Big) = -4.$$

5. *Assume $a_r \in \mathbb{F}_{2^{\frac{m}{2}}}$. If $b' \in \{\beta+\beta^2, \beta+\beta^3, \beta^2+\beta^4, \beta^3+\beta^4, \beta+\beta^4, \beta^2+\beta^3\}$ where β is a primitive 5-th root of unity in $\mathbb{F}_{16}$ then, $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if,*

$$\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(D_5(x))\Big) = 2.$$

Proof. Let us study the semi-bentness of $\tilde{f}_{a_r,b',c}$ for $b' \in \mathbb{F}_{16}$. Thanks to [19], Condition 2) of Corollary 1 (that is, in this case : $\sum_{u\in U} \chi\Big(\tilde{g}_{a_r,b'}(u)\Big) = 1$ where $\tilde{g}_{a_r,b'} := \sum_{r\in R} Tr_1^n(a_r x^{r(2^m-1)}) + Tr_1^2(bx^{\frac{2^n-1}{5}})$) is satisfied if and only if Condition 1) (resp. Condition 2), Condition 3), Condition 4), Condition 5)) of the proposition holds in the case where b' a primitive element of $\mathbb{F}_{16}$ such that $Tr_1^4(b') = 0$ (resp. $b' = 1$, $b' \in \{\beta, \beta^2, \beta^3, \beta^4\}$, b' is a primitive element of $\mathbb{F}_{16}$ such that $Tr_1^4(b') = 1$, $b' \in \{\beta+\beta^2, \beta+\beta^3, \beta^2+\beta^4, \beta^3+\beta^4, \beta+\beta^4, \beta^2+\beta^3\}$ where β is a primitive 5-th root of unity in $\mathbb{F}_{16}$). On the other hand, the number of solutions u in U of the equation $Tr_m^n(\omega u) + c^{\frac{1}{2}} = 0$ is either 0 or 2 for every $w \in \mathbb{F}_{2^n}$ (see proof Proposition of[1]), which proves the semi-bentness property of $\tilde{f}_{a_r,b',c}$, according to Corollary 1. For the semi-bentness of $\tilde{f}'_{a_r,b'}$, we use the same arguments as previously and the fact that the number of solutions u in U of the equation $Tr_m^n(wu) + Tr_m^n(u^{2s'-1}) = 0$ with $s' = 3$ (resp. $s' = 1/6$ where m even) is either 0 or 2 for every $w \in \mathbb{F}_{2^n}$ (according to Dobbertin et al. [6]).

Remark 2. Let β be a primitive element of $\mathbb{F}_{16}$ such that $Tr_1^4(\beta)=0$ and $c\in\mathbb{F}_{2^m}^\star$. Note that for every positive integer d such that d is co-prime with $\frac{2^m+1}{5}$, the function defined on $\mathbb{F}_{2^n}$ by $x\mapsto \sum_{r\in R} Tr_1^n(a_r x^{dr(2^m-1)})+Tr_1^4(\beta x^{\frac{2^n-1}{5}})+Tr_1^m(cx^{2^m+1})$ is also semi-bent if and only if the Condition 1) of Theorem 2 holds, according to Proposition 3.17 [19].

4 Efficient Characterizations of Semi-bent Functions with Multiple Trace Terms by Means of Cardinalities of Hyperelliptic Curves

Proposition 1 and Proposition 2 provide a test of semi-bentness for the functions $f_{a_r,b,c}$ (resp. $f'_{a_r,b}$) of the form (1) (resp. form (2)) and for functions $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) of the form (3) (resp. form (4)) with exponential complexity. Indeed, suppose R is fixed and m is variable then, for any given sequences of coefficients $a_r\in\mathbb{F}_{2^m}^\star$ (where $r\in R$), checking whether Condition 1 and Condition 2 of Proposition 1 and Proposition 2 are satisfied requires time that is exponential in m (hence it also exponential in n). The aim of this section is to exhibit an efficient characterization of the semi-bentness of such functions. In the following, we shall use the hyperelliptic curve formalism to reduce computational complexity. We will show that semi-bent functions $f_{a_r,b,c}$, $f'_{a_r,b}$, $\tilde{f}_{a_r,b',c}$ and $\tilde{f}'_{a_r,b'}$ can be described in terms of cardinalities of some hyperelliptic curves. To this end, we need some background on hyperelliptic curves as well as results about point counting on such curves over finite fields of characteristic 2.

4.1 Point Counting on Algebraic Curves

In this subsection we give briefly basic definitions for hyperelliptic curves as well as results about point counting on such curves over finite fields of characteristic 2. Given a curve E defined on $\mathbb{F}_{2^m}$, $\#E$ means the number of points on it with coordinates in the given finite field $\mathbb{F}_{2^m}$. The fact about such curves we will use in the next section is that there exist algorithms to compute their cardinalities in polynomial time and space in m. A description of the different types of hyperelliptic curves in even characteristic can be found in [7]. For the cryptographic point of view, the curves are often chosen to be imaginary hyperelliptic curves. This is also the kind of curves we will encounter. Such a hyperelliptic curve of genus g can be described by an affine part given by the following equation:

$$H: y^2+h(x)y=f(x),$$

where $h(x)$ is of degree $\leq g$ and $f(x)$ is monic of degree $2g+1$.

The main result about point counting of hyperelliptic curves we use is given by Vercauteren [18].

Theorem 1. *(Theorem 4.4.1 page 135, [18]) Let H be a hyperelliptic curve of genus g defined over $\mathbb{F}_{2^m}$. There exists an algorithm to compute the cardinality of H in*

$$O(g^3m^3(g^2+\log^2 m\log\log m)\log gm\log\log gm)$$

bit operations and $O(g^4m^3)$ *memory.*

A stronger result in terms of spaces is also given for hyperelliptic curves of a special form.

Definition 3. *An Artin-Schreier curve is a hyperelliptic curve whose affine part is given by an equation of the form:*

$$H : y^2 + x^n y = f(x),$$

where $0 \leq n \leq g$ *and* $f(x)$ *is monic of degree* $2g+1$.

Theorem 2. *(Theorem 4.3.1 page 114, [18]) Let* H *be an Artin-Schreier curve of genus* g *defined over* $\mathbb{F}_{2^m}$. *There exists an algorithm to compute the cardinality of* H *in*

$$O(g^3m^3(g^2+\log^2 m\log\log m)\log gm\log\log gm)$$

bit operations and $O(g^3m^3)$ *memory.*

Using complex methods involving deformations theory, better complexities were obtained recently thanks to Hubrechts's result [9].

4.2 Characterizations of Semi-bentness in Terms of Cardinalities of Hyperelliptic Curves

The criteria of semi-bentness given by Proposition 1 and Proposition 2 lead to an exponential time and space algorithm. In the following, we shall use the hyperelliptic curve formalism to reduce computational complexity. The characterizations of semi-bentness given by Proposition 1 can be reformulated in terms of cardinality of hyperelliptic curves as follows.

Theorem 3. *The notations are as in Proposition 1. Moreover, let* $H_{a_r}^{(1)}$, $H_{a_r}^{(2)}$ *and* $H_{a_r}^{(3)}$ *be the (affine) curves defined over* $\mathbb{F}_{2^m}$ *by*

$$H_{a_r}^{(1)} : y^2 + y = \sum_{r\in R} a_r D_r(x),$$
$$H_{a_r}^{(2)} : y^2 + y = \sum_{r\in R} a_r D_r(x+x^3).$$
$$H_{a_r}^{(3)} : y^2 + xy = x + x^2\sum_{r\in R} a_r D_r(x),$$

a) If β *is a primitive element of* $\mathbb{F}_4$, *then* $f_{a_r,\beta,c}$ *(resp.* $f'_{a_r,\beta}$*) is semi-bent if and only if*

$$2\#H_{a_r}^{(2)} - \left(\#H_{a_r}^{(1)} + \#H_{a_r}^{(3)}\right) = -3.$$

b) If $b = 1$, *then* $f_{a_r,1,c}$ *(resp.* $f'_{a_r,1}$*) is semi-bent if and only if*

$$4\#H_{a_r}^{(2)} - 5\#H_{a_r}^{(1)} + \#H_{a_r}^{(3)} = 3.$$

Proof. a) According to Proposition 1, $f_{a_r,\beta,c}$ (resp. $f'_{a_r,\beta}$) is semi-bent if and only if, $f_{a_r,\beta^2,c}$ (resp. f'_{a_r,β^2}) is semi-bent i.e.

$$\sum_{x\in\mathbb{F}_{2^m}} \chi\Big(Tr_1^m(x^{-1}) + h_{a_r}(D_3(x))\Big) = 2^m - 2wt(h_{a_r}\circ D_3) + 4. \tag{6}$$

Therefore, using the fact that, for a Boolean function f defined on $\mathbb{F}_{2^n}$, $\sum_{x\in\mathbb{F}_{2^n}}\chi(f(x)) = 2^n - 2wt(f)$ and that the indicator of the set $\{x\in\mathbb{F}_{2^m}^\star \mid Tr_1^m(x^{-1})=1\}$ can be written as $\frac{1}{2}\left(1-\chi(Tr_1^m(x^{-1}))\right)$, we get that Condition (6) is equivalent to $\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(D_3(x))\Big) = -2$. Now, since m is odd, $Tr_1^m(1)=1$. Moreover, for $c\in\mathbb{F}_{2^m}^*$, the equation $x^3+x=c$ has a unique solution in $\mathbb{F}_{2^m}$ if and only if $Tr_1^m(\frac{1}{c^2}) \neq Tr_1^m(1)=1$ i.e $Tr_1^m(c^{-1})=0$. Therefore, the mapping $x\mapsto D_3(x) := x^3+x$ is a permutation of the set $\{x\in\mathbb{F}_{2^m}^* \mid Tr_1^m(x^{-1})=0\}$. Thus, $\{x^3+x \mid x\in\mathbb{F}_{2^m}^*, Tr_1^m(x^{-1})=0\} = \{x\in\mathbb{F}_{2^m}^*, Tr_1^m(x^{-1})=0\}$ ($Tr_1^m(\frac{1}{x^3+x}) = Tr_1^m(x^{-1})$ for $x\in\mathbb{F}_{2^m}^*\setminus\{1\}$). The next equalities hold.

$$\begin{aligned}
\sum_{x\in\mathbb{F}_{2^m}^*, Tr_1^m(x^{-1})=1} \chi\left(h_{a_r}\circ D_3(x)\right) &= \sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(h_{a_r}\circ D_3(x)\right) - \sum_{x\in\mathbb{F}_{2^m}^*, Tr_1^m(x^{-1})=0} \chi\left(h_{a_r}\circ D_3(x)\right)\\
&= \sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(h_{a_r}\circ D_3(x)\right) - \sum_{x\in\mathbb{F}_{2^m}^*, Tr_1^m(x^{-1})=0} \chi\left(h_{a_r}(x)\right)\\
&= \sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(h_{a_r}\circ D_3(x)\right)\\
&\quad \frac{1}{2}\left(\sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(h_{a_r}(x)\right) + \sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(Tr_1^m(x^{-1}) + h_{a_r}(x)\right)\right).
\end{aligned}$$

Now, firstly, we have $\sum_{x\in\mathbb{F}_{2^m}^*}\chi\left(h_{a_r}\circ D_3(x)\right) = -2^m - 1 + \#H_{a_r}^{(2)}$. Indeed

$$\begin{aligned}
\sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(h_{a_r}\circ D_3(x)\right) &= 2^m - 1 - 2\#\{x\in\mathbb{F}_{2^m}^* \mid h_{a_r}\circ D_3(x)=1\}\\
&= 2^m - 1 - 2(2^m - \#\{x\in\mathbb{F}_{2^m} \mid h_{a_r}\circ D_3(x)=0\}
\end{aligned}$$

Using the additive version of Hillbert 90 : $Tr_1^m(x)=0 \iff \exists t\in\mathbb{F}_{2^m}, x=t^2+t$ we get,

$$\begin{aligned}
\sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(h_{a_r}\circ D_3(x)\right) &= -2^m - 1 + 2\#\{x\in\mathbb{F}_{2^m} \mid \exists t\in\mathbb{F}_{2^m}, t^2+t = \sum_{r\in R} a_r D_r(D_3(x))\}\\
&= -2^m - 1 + \#H_{a_r}^{(2)}.
\end{aligned}$$

Secondly, we have $\sum_{x\in\mathbb{F}_{2^m}^*}\chi\left(h_{a_r}(x)\right)=-2^m-1+\#H_{a_r}^{(1)}$. Indeed,

$$\begin{aligned}\sum_{x\in\mathbb{F}_{2^m}^*}\chi\left(h_{a_r}(x)\right)&=2^m-1-2\#\{x\in\mathbb{F}_{2^m}^*\mid h_{a_r}(x)=1\}\\&=2^m-1-2(2^m-\#\{x\in\mathbb{F}_{2^m}\mid h_{a_r}(x)=0\}\\&=-2^m-1+2\#\{x\in\mathbb{F}_{2^m}\mid \exists t\in\mathbb{F}_{2^m}, t^2+t=\sum_{r\in R}a_rD_r(x)\}\\&=-2^m-1+\#H_{a_r}^{(1)}.\end{aligned}$$

Thirdly, we have $\sum_{x\in\mathbb{F}_{2^m}^*}\chi\left(Tr_1^m(x^{-1})+h_{a_r}(x)\right)=-2^m+\#H_{a_r}^{(3)}$. Indeed,

$$\begin{aligned}\sum_{x\in\mathbb{F}_{2^m}^*}\chi\left(Tr_1^m(x^{-1})+h_{a_r}(x)\right)&=\sum_{x\in\mathbb{F}_{2^m}^*}(1-2(Tr_1^m(x^{-1})+h_{a_r}(x)))\\&=2^m-1-2\#\{x\in\mathbb{F}_{2^m}^*\mid Tr_1^m(x^{-1})+h_{a_r}(x)=1\}\\&=-2^m+1+2\#\{x\in\mathbb{F}_{2^m}^*\mid \exists t\in\mathbb{F}_{2^m}\, t^2+t=x^{-1}+\sum_{r\in R}a_rD_r(x)\}\\&=-2^m+1+2\#\{x\in\mathbb{F}_{2^m}^*\mid \exists t\in\mathbb{F}_{2^m}\, t^2+xt=x+x^2\sum_{r\in R}a_rD_r(x)\}\\&=-2^m+1+\#H_{a_r}^{(3)}-\#\{P\in H_{a_r}^{(3)}\mid x=0\}=-2^m+\#H_{a_r}^{(3)}.\end{aligned}$$

We finally obtain,

$$\sum_{x\in\mathbb{F}_{2^m}^*\mid Tr_1^m(x^{-1})=1}\chi\left(h_{a_r}\circ D_3(x)\right)=\left(-2^m-1+\#H_{a_r}^{(2)}\right)-\frac{1}{2}\left(\left(-2^m-1+\#H_{a_r}^{(1)}\right)+\left(-2^m+\#H_{a_r}^{(3)}\right)\right)$$

that is,

$$\sum_{x\in\mathbb{F}_{2^m}^*\mid Tr_1^m(x^{-1})=1}\chi\left(h_{a_r}\circ D_3(x)\right)=-\frac{1}{2}+\#H_{a_r}^{(2)}-\frac{1}{2}\left(\#H_{a_r}^{(1)}+\#H_{a_r}^{(3)}\right).\quad(7)$$

The assertion 1) follows.

b) According to Theorem 1, $f_{a_r,1,c}$ (resp. $f'_{a_r,1}$) is semi-bent if and only if,

$$\begin{aligned}&3\sum_{x\in\mathbb{F}_{2^m}}\chi\Big(Tr_1^m(x^{-1})+h_{a_r}(x)\Big)-2\sum_{x\in\mathbb{F}_{2^m}}\chi\Big(Tr_1^m(x^{-1})+h_{a_r}(D_3(x))\Big)\\&=4+2^m+4wt(h_{a_r}\circ D_3)-6wt(h_{a_r}).\end{aligned}$$

We have

$$\begin{aligned}\sum_{x\in\mathbb{F}_{2^m}}\chi(Tr_1^m(x^{-1})+h_{a_r}(x))&=\sum_{x\in\mathbb{F}_{2^m}}\chi(h_{a_r}(x))-2\sum_{x\in\mathbb{F}_{2^m}^*\mid Tr_1^m(x^{-1})=1}\chi(h_{a_r}(x))\\&=2^m-2wt(h_{a_r})-2\sum_{x\in\mathbb{F}_{2^m}^*\mid Tr_1^m(x^{-1})=1}\chi(h_{a_r}(x)).\end{aligned}$$

Similarly we have,

$$\sum_{x\in\mathbb{F}_{2m}} \chi(Tr_1^m(x^{-1}) + h_{a_r}(D_3(x)) = \sum_{x\in\mathbb{F}_{2m}} \chi(h_{a_r}(D_3(x)) - 2 \sum_{x\in\mathbb{F}_{2^m}^* | Tr_1^m(x^{-1})=1} \chi(h_{a_r}(D_3(x))$$
$$= 2^m - 2wt(h_{a_r} \circ D_3) - 2 \sum_{x\in\mathbb{F}_{2^m}^* | Tr_1^m(x^{-1})=1} \chi(h_{a_r}(D_3(x)).$$

The above necessary and sufficient condition of semi-bentness is then equivalent to

$$-3 \sum_{x\in\mathbb{F}_{2^m}^* | Tr_1^m(x^{-1})=1} \chi(h_{a_r}(x)) + 2 \sum_{x\in\mathbb{F}_{2^m}^* | Tr_1^m(x^{-1})=1} \chi(h_{a_r}(D_3(x)) = 2. \quad (8)$$

Now, according to previous calculations, we have $\sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(h_{a_r}(x)\right) = -2^m - 1 + \#H_{a_r}^{(1)}$ and $\sum_{x\in\mathbb{F}_{2^m}^*} \chi\left(Tr_1^m(x^{-1}) + h_{a_r}(x)\right) = -2^m + \#H_{a_r}^{(3)}$. Thus,

$$\sum_{x\in\mathbb{F}_{2^m}^* | Tr_1^m(x^{-1})=1} \chi(h_{a_r}(x)) = \frac{1}{2}(-2^m - 1 + \#H_{a_r}^{(1)}) - \frac{1}{2}(-2^m + \#H_{a_r}^{(3)})$$
$$= -\frac{1}{2} + \frac{1}{2}\#H_{a_r}^{(1)} - \frac{1}{2}\#H_{a_r}^{(3)}.$$

Finally, using relation (9), we obtain that condition (8) is equivalent to

$$-3\left(-\frac{1}{2} + \frac{1}{2}\#H_{a_r}^{(1)} - \frac{1}{2}\#H_{a_r}^{(3)}\right) + 2\left(-\frac{1}{2} + \#H_{a_r}^{(2)} - \frac{1}{2}(\#H_{a_r}^{(1)} + \#H_{a_r}^{(3)})\right) = 2$$

that is, $4\#H_{a_r}^{(2)} - 5\#H_{a_r}^{(1)} + \#H_{a_r}^{(3)} = 3$ which completes the proof.

Likewise, the characterizations of semi-bentness given by Proposition 2 can also be reformulated in terms of cardinalities of hyperelliptic curves as follows.

Theorem 4. *The notations are as in Proposition 2. Moreover, let* $H_{a_r}^{(1)}$, $H_{a_r}^{(3)}$, $\tilde{H}_{a_r}^{(2)}$ *and* $\tilde{H}_{a_r}^{(3)}$ *be the (affine) curves defined over* $\mathbb{F}_{2^m}$ *by*

$$H_{a_r}^{(1)} : y^2 + y = \sum_{r\in R} a_r D_r(x),$$
$$H_{a_r}^{(3)} : y^2 + xy = x + x^2 \sum_{r\in R} a_r D_r(x),$$
$$\tilde{H}_{a_r}^{(2)} : y^2 + y = \sum_{r\in R} a_r D_r(x + x^3 + x^5),$$
$$\tilde{H}_{a_r}^{(3)} : y^2 + xy = x + x^2 \sum_{r\in R} a_r D_r(x + x^3 + x^5).$$

1. *If Let b' a primitive element of $\mathbb{F}_{16}$ such that $Tr_1^4(b')=0$, then $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if,*
$$\#\tilde{H}_{a_r}^{(2)} - \#\tilde{H}_{a_r}^{(3)} = 5.$$
2. *If $b'=1$, then $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if*
$$2(\#\tilde{H}_{a_r}^{(2)} - \#\tilde{H}_{a_r}^{(3)}) - 5(\#H_{a_r}^{(1)} - \#H_{a_r}^{(3)}) = 5.$$
3. *Assume $a_r \in \mathbb{F}_{2^{\frac{m}{2}}}$. If $b' \in \{\beta, \beta^2, \beta^3\beta^4\}$ where β is a primitive 5-th root of unity in $\mathbb{F}_{16}$, then $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if,*
$$\#\tilde{H}_{a_r}^{(2)} - \#\tilde{H}_{a_r}^{(3)} + 5(\#H_{a_r}^{(1)} - \#H_{a_r}^{(3)}) = -10.$$
4. *Assume $a_r \in \mathbb{F}_{2^{\frac{m}{2}}}$. If b' is a primitive element of $\mathbb{F}_{16}$ such that $Tr_1^4(b')=1$, then $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) if and only if,*
$$3\Big(\#\tilde{H}_{a_r}^{(2)} - \#\tilde{H}_{a_r}^{(3)}\Big) + 5\Big(\#H_{a_r}^{(3)} - \#H_{a_r}^{(1)}\Big) = -10.$$
5. *Assume $a_r \in \mathbb{F}_{2^{\frac{m}{2}}}$. If $b' \in \{\beta+\beta^2, \beta+\beta^3, \beta^2+\beta^4, \beta^3+\beta^4, \beta+\beta^4, \beta^2+\beta^3\}$ where β is a primitive 5-th root of unity in $\mathbb{F}_{16}$, then $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if,*
$$\#\tilde{H}_{a_r}^{(2)} - \#\tilde{H}_{a_r}^{(3)} = 5.$$

Proof. 1. According to Proposition 2, the function $\tilde{f}_{a_r,b',c}$ (resp. $\tilde{f}'_{a_r,b'}$) is semi-bent if and only if
$$\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(D_5(x))\Big) = 2.$$
The sum $\sum_{x\in\mathbb{F}_{2^m}^\star Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(D_5(x))\Big)$ can be expressed in terms of cardinalities of the hyperelliptic curves $\tilde{H}_{a_r}^{(2)}$ and $\tilde{H}_{a_r}^{(3)}$ as follows: we have
$$\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1} \chi\Big(h_{a_r}(D_5(x))\Big) = \frac{1}{2}\Big(\sum_{x\in\mathbb{F}_{2^m}^\star} \chi\Big(h_{a_r}(D_5(x))\Big) - \sum_{x\in\mathbb{F}_{2^m}^\star} \chi(Tr_1^m(x^{-1}) + h_{a_r}(D_5(x)))\Big).$$
Now, on one hand, we have (using the additive version of Hillbert 90)
$$\begin{aligned}\sum_{x\in\mathbb{F}_{2^m}^\star} \chi\Big(h_{a_r}(D_5(x))\Big) &= 2^m - 2wt(h_r(D_5(x))) - 1\\ &= -2^m - 1 + 2\#\{x\in\mathbb{F}_{2^m}, Tr_1^m\Big(\sum_{r\in R} a_r D_r\circ D_5(x)\Big) = 0\}\\ &= -2^m - 1 + 2\#\{x\in\mathbb{F}_{2^m}, \exists t\in\mathbb{F}_{2^m}, \sum_{r\in R} a_r D_r\circ D_5(x) = t+t^2\}\\ &= -2^m - 1 + \#H_{a_r}^{(2)}.\end{aligned}$$

On the other hand,

$$\begin{aligned}\sum_{x\in\mathbb{F}_{2^m}^\star}\chi(Tr_1^m(x^{-1})+h_{a_r}(x))\Big) &= \sum_{x\in\mathbb{F}_{2^m}^\star}\Big(1-2(Tr_1^m(x^{-1})+h_{a_r}(x))\Big)\\ &=-2^m+1+2\#\{x\in\mathbb{F}_{2^m}^\star,\exists t\in\mathbb{F}_{2^m},x^{-1}+\sum_{r\in R}a_rD_r(x)=t+t^2\}\\ &=-2^m+1+2\#\{x\in\mathbb{F}_{2^m}^\star,\exists t\in\mathbb{F}_{2^m},x+x^2\sum_{r\in R}a_rD_r(x)=tx+t^2\}\\ &=-2^m+1+\#\tilde{H}_{a_r}^{(3)}-\#\{P\in\tilde{H}_{a_r}^{(3)},x=0\}=-2^m+\#\tilde{H}_{a_r}^{(3)}.\end{aligned}$$

Collecting the calculations of the two sums, we obtain:

$$\sum_{x\in\mathbb{F}_{2^m}^\star,Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(D_5(x))\Big)=\frac{1}{2}\Big((2^m-1+\#\tilde{H}_{a_r}^{(2)})-(2^m+\#\tilde{H}_{a_r}^{(3)})\Big)$$

that is,

$$\sum_{x\in\mathbb{F}_{2^m}^\star,Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(D_5(x))\Big)=\frac{1}{2}\Big(-1+\#\tilde{H}_{a_r}^{(2)}-\#\tilde{H}_{a_r}^{(3)}\Big). \quad (9)$$

The result follows.

2. According to Theorem 2, the function $\tilde{f}_{a_r,1,c}$ (resp. $\tilde{f}'_{a_r,1}$) is semi-bent if and only if,

$$2\sum_{x\in\mathbb{F}_{2^m}^\star,Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(D_5(x))\Big)-5\sum_{x\in\mathbb{F}_{2^m}^\star,Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(x)\Big)=4.$$

We have already the expression of the sum $\sum_{x\in\mathbb{F}_{2^m},Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(D_5(x))\Big)$ in terms of cardinalities of the curve $\tilde{H}_{a_r}^{(2)}$ and $\tilde{H}_{a_r}^{(3)}$. It remains to express the sum $\sum_{x\in\mathbb{F}_{2^m},Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(x)\Big)$ in terms of cardinal of hyperelliptic curves. To this end, we proceed in the same way:

$$\sum_{x\in\mathbb{F}_{2^m}^\star,Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(x)\Big)=\frac{1}{2}\Big(\sum_{x\in\mathbb{F}_{2^m}^\star}\chi\Big(h_{a_r}(x)\Big)-\sum_{x\in\mathbb{F}_{2^m}^\star}\chi(Tr_1^m(x^{-1})+h_{a_r}(x))\Big).$$

Now, on one hand, we have

$$\sum_{x\in\mathbb{F}_{2^m}^\star}\chi\Big(h_{a_r}(x)\Big) = 2^m - 2wt(h_r(x)) - 1$$

$$= -2^m - 1 + 2\#\{x\in\mathbb{F}_{2^m}, Tr_1^m\Big(\sum_{r\in R} a_r D_r(x)\Big) = 0\}$$

$$= -2^m - 1 + 2\#\{x\in\mathbb{F}_{2^m}, \exists t\in\mathbb{F}_{2^m}, \sum_{r\in R} a_r D_r(x) = t + t^2\} = -2^m - 1 + \#H_{a_r}^{(1)}.$$

On the other hand,

$$\sum_{x\in\mathbb{F}_{2^m}^\star}\chi(Tr_1^m(x^{-1}) + h_{a_r}(x)\Big) = \sum_{x\in\mathbb{F}_{2^m}^\star}\Big(1 - 2(Tr_1^m(x^{-1}) + h_{a_r}(x))\Big)$$

$$= -2^m + 1 + 2\#\{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1} + \sum_{r\in R} a_r D_r(x) = 0\}$$

$$= -2^m + 1 + 2\#\{x\in\mathbb{F}_{2^m}^\star, \exists t\in\mathbb{F}_{2^m}, x^{-1} + \sum_{r\in R} a_r D_r(x) = t + t^2\}$$

$$= -2^m + 1 + 2\#\{x\in\mathbb{F}_{2^m}^\star, \exists t\in\mathbb{F}_{2^m}, x + x^2\sum_{r\in R} a_r D_r(x) = tx + t^2\}$$

$$= -2^m + 1 + \#H_{a_r}^{(3)} - \#\{P\in\tilde{H}_{a_r}^{(3)}, x = 0\} = -2^m + \#H_{a_r}^{(3)}.$$

Collecting the calculations of the two sums, we obtain:

$$\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(x)\Big) = \frac{1}{2}\Big(-2^m - 1 + \#H_{a_r}^{(1)} - (-2^m + \#H_{a_r}^{(3)})\Big)$$

$$\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(x)\Big) = \frac{1}{2}\Big(-1 + \#H_{a_r}^{(1)} - \#H_{a_r}^{(3)}\Big) \tag{10}$$

Now, the function $\tilde{f}_{a_r,1,c}$ (resp. $\tilde{f}'_{a_r,1}$) is semi-bent if and only if,

$$2\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(D_5(x))\Big) - 5\sum_{x\in\mathbb{F}_{2^m}^\star, Tr_1^m(x^{-1})=1}\chi\Big(h_{a_r}(x)\Big) = 4.$$

that is,

$$\Big(-1 + \#\tilde{H}_{a_r}^{(2)} - \#\tilde{H}_{a_r}^{(3)}\Big) - \frac{5}{2}\Big(-1 + \#H_{a_r}^{(1)} - \#H_{a_r}^{(3)}\Big) = 4.$$

The result follows. Assertions 3), 4) and 5) follow from Proposition 2, and relations (9) and (10).

4.3 Advantage of the Reformulation and Experimental Results

Let the set R be fixed. Let r_{max} be the maximal index in R. One can assume that r_{max} is odd for two reasons. The first reason is that r_{max} should be as small as possible for efficiency reasons so the natural choice for the indices in a cyclotomic coset will be the cost leaders which are odd numbers. The second reason is that if r_{max} is supposed to be odd then the curves $H_{a_r}^{(1)}$, $H_{a_r}^{(2)}$, $H_{a_r}^{(3)}$, $\tilde{H}_{a_r}^{(2)}$ and $\tilde{H}_{a_r}^{(3)}$ are necessary Artin-Schreir curves. Theorem 1, Theorem 2 and Theorem 2 in [9] show that there exist efficient algorithms to compute the cardinality of such curves as long as r_{max} is supposed to be relatively small. The complexity of point counting algorithms depends on the genera of the curves and so on the degree of the polynomials involved to define them. Table 1 gives the degree of the polynomials which define the curves $H_{a_r}^{(1)}$, $H_{a_r}^{(2)}$, $H_{a_r}^{(3)}$, $\tilde{H}_{a_r}^{(2)}$ and $\tilde{H}_{a_r}^{(3)}$ as well as the genus of each curve.

Table 1. Genus and degree of the curves

Curve	degree of the polynomial defining the curve	genus
$H_{a_r}^{(1)}$	r_{max}	$\frac{r_{max}-1}{2}$
$H_{a_r}^{(2)}$	$3r_{max}$	$\frac{3r_{max}-1}{2}$
$H_{a_r}^{(3)}$	$r_{max}+2$	$\frac{r_{max}+1}{2}$
$\tilde{H}_{a_r}^{(2)}$	$5r_{max}$	$\frac{5r_{max}-1}{2}$
$\tilde{H}_{a_r}^{(3)}$	$5r_{max}+2$	$\frac{5r_{max}+1}{2}$

From [9], we deduce the following result which gives the complexity of the semi-bentness test obtained thanks to Theorem 3 and Theorem 4.

Theorem 5. *The notations are as in Proposition 1 and Proposition 2. Let r_{max} be the maximal index in R. Then, the semi-bentness of $f_{a_r,b,c}$, $f'_{a_r,b}$, $\tilde{f}_{a_r,b',c}$ and $\tilde{f}'_{a_r,b'}$ can be checked in $O(r_{max}^{7.376}m^2 + r_{max}^{3.376}m^{2.667})$ bit operations and $O(r_{max}^5 m^2 + r_{max}^3 m^{2.5})$ memory.*

Therefore, if R is supposed to be fixed, then so are r_{max} and the genera of the curves thus the complexity of the semi-bentness test (using the reformulation in terms of cardinalities of hyperelliptic curves) is polynomial in m. Asymptotically, this is much better than a straightforward application of semi-bentness where the exponential sums on $\mathbb{F}_{2^m}$ are naively computed one term at a time. It should be remarked that if no restriction is cast upon R, then the maximal index r_{max} will obviously depend on m and will in fact grow. Nonetheless, fixing a set R, that is, only looking for Boolean functions with a given polynomial form within a large family, is customary in cryptographic applications. We have studied the practical impact of the reformulations given by Theorem 3 and Theorem 4. More precisely, we have performed several simulations with Magma V2-17-13 (for our computations, we have chosen the sets $R = \{1\}$ and $R = \{1, 3\}$ of indices and some couples of coefficients (a_1, a_3) randomly generated in $\mathbb{F}_{2^m}^{\star}$) and found

that the practical difference between the two reformulations is non-negligible. Indeed, the timings of computing exponential sums (leading to semi-bentness test for functions of Class (1) and Class (2) with $R = 1$ (resp. $R = \{1, 3\}$) using Proposition 1) is much longer than computing cardinalities of hyperelliptic curves using Theorem 3).

Acknowledgement. The author thanks Jean-Pierre Flori for his interesting discussion.

References

1. Canteaut, A., Carlet, C., Charpin, P., Fontaine, C.: On cryptographic properties of the cosets of R(1,m). IEEE Transactions on Information Theory 47, 1494–1513 (2001)
2. Carlet, C., Mesnager, S.: On Semi-bent Boolean functions. IEEE Transactions on Information Theory 58(5), 3287–3292 (2012)
3. Charpin, P., Pasalic, E., Tavernier, C.: On bent and semi-bent quadratic Boolean functions. IEEE Transactions on Information Theory 51(12), 4286–4298 (2005)
4. Chee, S., Lee, S., Kim, K.: Semi-bent Functions. In: Safavi-Naini, R., Pieprzyk, J.P. (eds.) ASIACRYPT 1994. LNCS, vol. 917, pp. 107–118. Springer, Heidelberg (1995)
5. Cheon, J.H., Chee, S.: Elliptic Curves and Resilient Functions. In: Won, D. (ed.) ICISC 2000. LNCS, vol. 2015, pp. 64–397. Springer, Heidelberg (2000)
6. Dobbertin, H., Leander, G., Canteaut, A., Carlet, C., Felke, P., Gaborit, P.: Construction of bent functions via Niho Power Functions. Journal of Combinatorial therory, Serie A 113, 779–798 (2006)
7. Enge, A.: How to distinguish hyperelliptic curves in even characteristic. In: Public-Key Cryptography and Computational Number Theory. de Gruyter Proceedings in Mathematics. DE GRUYTER (2011)
8. Flori, J., Mesnager, S.: An efficient characterization of a family of hyperbent functions with multiple trace terms (in preprint)
9. Hubrechts, H.: Point counting in families of hyperelliptic curves in characteristic 2. LMS J. Comput. Math. 10, 207–234 (2007)
10. Lisoněk, P.: An efficient characterization of a family of hyperbent functions. IEEE Transactions on Information Theory 57, 6010–6014 (2011)
11. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
12. Meier, W., Staffelbach, O.: Fast Correlation Attacks on Stream Ciphers. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 301–314. Springer, Heidelberg (1988)
13. Mesnager, S.: Hyper-bent Boolean Functions with Multiple Trace Terms. In: Hasan, M.A., Helleseth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 97–113. Springer, Heidelberg (2010)
14. Mesnager, S.: A new class of bent and hyper-bent Boolean functions in polynomial forms. Des. Codes Cryptography 59, 265–279 (2011)
15. Mesnager, S.: Semi-bent functions from Dillon and Niho exponents, Kloosterman sums and Dickson polynomials. IEEE Transactions on Information Theory 57(11), 7443–7458 (2011)

16. Rothaus, O.S.: On "bent" functions. J. Combin.Theory Ser. A 20, 300–305 (1976)
17. Sun, G., Wu, C.: Construction of Semi-Bent Boolean Functions in Even Number of Variables. Chinese Journal of Electronics 18(2) (2009)
18. Vercauteren, F.: Computing zeta functions of curves over finite fields. PhD thesis, Katholieke Universiteit Leuven (2003)
19. Wang, B., Tang, C., Qi, Y., Yang, Y., Xu, M.: A New Class of Hyper-bent Boolean Functions with Multiple Trace Terms. In: Cryptology ePrint Archive, Report 2011/600 (2011), http://eprint.iacr.org/
20. Zheng, Y., Zhang, X.-M.: Plateaued Functions. In: Varadharajan, V., Mu, Y. (eds.) ICICS 1999. LNCS, vol. 1726, pp. 284–300. Springer, Heidelberg (1999)

Complete Atomic Blocks for Elliptic Curves in Jacobian Coordinates over Prime Fields

Rodrigo Abarzúa[1,⋆] and Nicolas Thériault[2,⋆⋆]

[1] Institute of Computing, University of Campinas,
Av. Albert Einstein 1251, Campinas, Brazil
rabarzua@ic.unicamp.br
[2] Departamento de Matemática, Universidad del Bío-Bío,
Avda. Collao 1202, Concepción, Chile
ntheriau@ubiobio.cl

Abstract. In this paper we improve the safety aspects of previously published atomic blocks. We build new sets of atomic blocks designed to protect against both simple side-channel attacks and C-safe fault attacks for scalar multiplication for elliptic curves over prime fields. These atomic blocks are structured with the sequence of field operations (S, N, A, A, M, A), *Squaring, Negation, Addition, Addition, Multiplication, Addition*. We apply these atomic blocks to various operations in Jacobian coordinates: doubling, tripling, and quintupling, as well as mixed Jacobian-affine addition. We also give formulae for the general Jacobian addition for use in right-to-left scalar multiplication. Finally, we show how these techniques can be used to unify the Jacobian doubling formula with mixed Jacobian-affine addition, so they use the same number of atomic blocks.

Like previous atomic blocks formulae, our group operations provide protection against simple side channel attacks by dividing the group operations into smaller sequences of field operations. One of the main differences with our formulae resides in their security against C-safe fault attacks. Unlike previous works, our formulae are designed to completely fill the atomic blocks with field operations that affect the final output (i.e. we avoid "dummy" operations) and are all distinct (none of the operations are repeated). They also have the added bonus of being slightly more "compact" than most previous atomic blocks, having fewer additions/negations for each multiplication/squaring, potentially giving a performance gain.

Keywords: Elliptic curve, side-channel attack, C-safe fault attack, atomic blocks.

⋆ This research was funded by the Postdoctoral Fellowship Conicyt # 74110013.
⋆⋆ Research for this paper was supported in parts by FONDECYT (Chile) grant # 1110578 and by the Anillo project ACT 56 (CONICYT, Chile).

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 37–55, 2012.

1 Introduction

Elliptic Curves Cryptography (ECC) is a public-key cryptosystem proposed by Neal Koblitz [35] and Victor Miller [42] in 1985 which provides significant advantages in several situations, including implementations on specialized microprocessors. For example, some industry standards require 1024-bits for the size of integers in the RSA system, whereas the equivalent requirement for ECC is to work with finite fields of 160-bits. Given the restrictions on embedded microprocessors (used in mobile devices), the ECC system is an interesting option obtain the required security.

Side-channel attacks exploit physical leakages of a cryptographic process on a device (using timing [36], power consumption [37] and electromagnetic radiation [46,24]). These attacks present a realistic threat to cryptographic applications, and have been demonstrated to be very effective against smart cards without proper countermeasures. There are two general strategies to these attacks: *Simple Side-channel Analysis* (SSCA) [36] which analyses the measurements of a single scalar multiplication, observing the differences in behavior of the scalar multiplication depending on the value of the secret key; and *Differential Side-channel Analysis* (DSCA) [37], which uses statistical techniques to retrieve information about the secret key based on the measurements from several scalar multiplications. In this paper we will focus mostly on SSCA.

Several proposals have been made to protect scalar multiplication against these attacks. For example, the *double-and-add-always* algorithm of Coron [20] ensures the sequence of operations to compute a scalar multiplication is independent of the value of the secret scalar by inserting a dummy point additions between consecutive doublings (when the bit of the scalar is 0). A second countermeasure is to use *unified formula* which use similar sets of field operations for both the general group additions and doublings. Such formulae exist for Edwards curves [22], inverted Edwards curves [6], curves in the Huff model [33], Hessian curves [48], Jacobi curves [38,10], Weierstrass elliptic curves [11] (more details can be found in the database of special elliptic curves [7]). Another possible countermeasure is the *Montgomery ladder* [45] designed for a special type of curves in large characteristic. As for the double-and-add-always algorithm, it makes sure that every bit of the scalar corresponds to both a doubling and an addition, but with the supplementary condition that both operations have an impact on the final output of the scalar multiplication. This was later generalized to all elliptic curves [41,11,28], and right-to-left scalar multiplication [34]. A fourth approach consists in using "regular" representations of the scalar [44,49,31], with the same fixed sequence of group operations for all scalars. Finally, Side-Channel Atomicity (first proposed by Chevallier-Mames *et al.* [16]) splits point operations into small homogeneous blocks of basic field operations. If it is carefully implemented, it becomes impossible to distinguish between atomic blocks that come from doublings or additions. Atomic blocks are potentially the most efficient SSCA countermeasure.

This paper is organized as follows. In the next section, we present state of the art for atomic blocks, a brief background on elliptic curves and the rescaling

methodology is presented in Section 3. Our techniques to secure atomic blocks against both SSCA and C-safe-fault attacks are presented in Section 4.1 and applied in Sections 4.2, 4.3 and 4.4. Finally, we compare the efficiency of our atomic blocks formulae to previous ones in Section 5.

2 State of the Art for Atomic Blocks

Atomic blocks formulae are a very promising method to secure scalar multiplication against SSCA. The idea was first introduced by Chevallier-Mames *et al.* [16] and consists in partitioning point operations into small homogeneous sequences of operations (atomic blocks) in such a way that the increase in cost when comparing against optimized formulae (without side-channel protection) is minimal in terms of operation counts, which cannot be distinguished from each other through SSCA, thus making it impossible for the attacker to know which block is part of a group doubling or addition. Any field operation of an atomic block that was not used by the formula would be filled with dummy operations (so no missing operation would be identified by a SSCA). Atomic block formula can the provide the desired security at a much lower price than other SSCA countermeasures.

Chen *et al.* in [15], presented an experimental attack on a smart card using an implementation of the atomic blocks proposed by Chevallier-Mames *et al.* [16]. This experimental attack utilized the different number of atomic blocks for group doublings and additions – for total operation times of $3.16ms$ and $3.61ms$ respectively – and a delay of $1.12ms$ for breaks between group operations. This experimental attacks was applied because the implementation does not avoid irregular breaks between atomic blocks within the same group operation and distinct group operations. Chen *et al.* proposed to balance the point doubling with respect to a group addition. A preferred option is to require a better management of the delays between atomic blocks, thus allowing for formulae with different numbers of blocks.

The original atomic block of Chevallier-Mames *et al.* had a structure of (M, A, N, A) (*Multiplication-Addition-Negation-Addition*) operations over the prime field. This atomic block made one important assumption: that multiplication and squaring are indistinguishable from a side-channel perspective. This was disproved by Amiel *et al.* [1] and Hanley *et al.* [30]. Since the Hamming weight for the results of a field multiplication and a squaring have different distributions, and the Hamming weight affects the side-channel traces. It is possible to use this difference to distinguish between blocks containing a general multiplication from those containing a squaring operation, re-opening the way to SSCA. As a consequence, atomic blocks had to consider distinct squaring and multiplication in their structure.

This distinction can also have some efficiency benefits since specialized squarings are less expensive than multiplication (at a ratio close to 0.8 in software practice [26]). To adapt existing formulae to various atomic block structures, the flexible methodology introduced by Longa and Miri [40], and Bernstein and

Lange [9] proves very useful. It permits the modification of point operations formulae to balance the number of squarings and multiplications, and thus facilitates the introduction of squarings into atomic blocks.

Longa and Miri presented a new atomic block structure based on the sequence *Squaring-Negation-Addition-Multiplication-Negation-Addition-Addition* of field operations – or (S, N, A, M, N, A, A). They applied their atomic block structure to doubling, tripling and mixed addition for elliptic curves in Jacobian coordinates. Giraud and Verneuil [26] presented a different atomic block structure for general Jacobian Addition and Modified Jacobian Doubling geared for right-to-left scalar multiplication, and in particular a scalar multiplication algorithm of Joye [32]. Their atomic blocks use the sequence of operations $(S, A, M, A, M, A, M, A, A, S, M, A, A, M, A, A, M, A)$. It should be noted that these atomic blocks formulae make use of dummy operations at one point or another – at the very least to fill up some of the additions and/or negations.

The Jacobian coordinates formulae of Longa and Miri [40] for left-to-right scalar multiplication and those of Giraud and Verneuil [26] for right-to-left scalar multiplication can be considered the current best in atomic blocks formulae.

Atomic blocks previously presented by the scientific community make use of false operations (additions and negations in the field). Since these operations have a computational cost (time, power consumption, etc.), which may be identified through a side-channel attacks.

As in the work of Yen [52], an attacker can inject a fault at specific times in the computation to see if certain operations are dummy or true, changing the value obtained during a specific field operation. If this was a false (dummy) operation, the changed output will not have an impact on the end-result of the scalar multiplication. On the other hand, if the operation is true, the fault impact on the final result of the scalar multiplication, producing an error. The valid/invalid output of the scalar multiplication can then be used to determine if the chosen field operation was dummy or true, and hence identify the group operation (addition, doubling, etc.) in use at that point of the computation. Through a relatively small number of such processes, an attacker can work out the secret key used, implying that the blocks previously presented ([16,40,26]) are not effective countermeasures to SSCA as they leave a weakness to C-safe fault attacks.

As a result, it is recommended to avoid using dummy operations in the formulae [4]. These attacks can be considered more closely related to DSCA than SSCA (although they are mathematically more simple than other DSCA), but they require far fewer observations than most other DSCA. Securing the atomic blocks formulae against C-safe fault attacks is the issue we address in our paper.

3 Mathematical Background

For a detailed description of elliptic curves, we refer the reader to [2].

An elliptic curve E defined over a large prime field $\mathbb{F}_p$ is given by an equation of the form $y^2 = x^3+ax+b$, with $4a^3+27b^2 \neq 0$. The group used for cryptography

consists of the (affine) points (x, y) on the curve and the point at infinity P_∞ (the neutral element), with the "chord-and-tangent" addition. The group operation for $(x_1, y_1) + (x_2, y_2)$ is given by

$$(x_3, y_3) = \left(\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1\right), \text{ where}$$

$$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } x_1 \neq x_2, \quad \textbf{[addition operation]} \\ \dfrac{3x_1^2 + a}{2y_1} & \text{if } (x_1, y_1) = (x_2, y_2). \quad \textbf{[doubling operation]} \end{cases}$$

and $(x_1, y_1) + (x_1, -y_1) = P_\infty$.

To an elliptic curve E defined by an equation of the form $E : y^2 = x^3 - 3x + b$ over a prime field $\mathbb{F}_p$, we associate the Jacobian projective curve $E_1 : Y^2 = X^3 - 3Z^4X + bZ^6$.

The (projective) points on E_1 are divided into classes by the equivalence relation:

$$(X_1, Y_1, Z_1) \equiv (X_2, Y_2, Z_2) \iff \exists \lambda \in \mathbb{F}_p^* \text{ s.t. } (X_2, Y_2, Z_2) = (\lambda^2 X_1, \lambda^3 Y_1, \lambda Z_1). \quad (1)$$

We denote $(X : Y : Z)$ the equivalence class containing the point (X, Y, Z). Note that every equivalence class with $Z \neq 0$ contains exactly one point of the form, $(x, y, 1)$ which corresponds to an (affine) point (x, y) on curve E (and vice versa). The (unique) equivalence class with $Z = 0$ is of the form $(\rho^3 : \rho^2 : 0)$ (with $\rho \in \mathbb{F}_p^*$) and corresponds to the point at infinity P_∞ of E.

Rescaling Methodology. The "rescaling" methodology presented by Longa and Miri [40] and Bernstein and Lange [9] takes advantage of the projective form of the point coordinates. The principal idea consists in taking a field multiplication $\alpha\beta$, and replacing it with a number of field squarings, additions and negations, via the equation

$$2\alpha\beta = (\alpha + \beta)^2 - \alpha^2 - \beta^2. \quad (2)$$

If we set $\lambda = 2$ in the class description (1), we easily see how factors of 2 can be incorporated into all of the coordinates (in our case, of the output of the computation). We can then replace the computation by (2) adjusting the remaining computations accordingly.

The technique presented in [40,9] can be summarized in two steps:

1. Replace one (or more) of the field multiplications by applying the algebraic substitution given in Equation (2).
2. Modify the point formula by inserting multiples of 2 in the point representation, using the equivalence $(X : Y : Z) \equiv (2^2X : 2^3Y : 2Z)$.

4 Atomic Blocks and C-Safe Attacks

As stated in the introduction, previously published atomic blocks formulae for elliptic curves defined over prime fields are open to C-safe fault attacks. Although

most equilibrated formulae do fill out all the multiplications and squarings with non-dummy operations, no such care is taken for field additions and negations.

Experimental data on various smart cards [26] give an addition-to-multiplication ratio close to 0.2 and a negation-to-multiplication ratio of 0.1. Even though the timings for these operations is much less than for multiplications and squarings (the squaring-to-multiplication ratio is usually close to 0.8), it would still be reasonable to mount a C-safe fault attack on dummy field additions and negations.

To address this weakness, we constructed new group operations formulae for point doublings, triplings and quintuplings in Jacobian coordinates as well as general Jacobian and mixed Jacobian-affine addition formulae.

4.1 Generating Complete Atomic Blocks

The only way to really avoid C-safe fault attacks is to ensure that every field operation of every atomic block is used in the computation of the final result. Note that it would not be sufficient to repeat the same operation more than once in the formula (using each result at least once), since the repeated operations would leave an essentially identical side-channel signature, thus re-opening the way to SSCA. We must therefore fill all the field operations of every atomic block, but always with different operands.

We used the following method to design our atomic blocks:

1. From the existing formulae, we determined that the most favorable form for the atomic blocks would be with $1S + 1M$ (since most formulae were close to equilibrated), with the squaring before the multiplication (due to the importance of squarings early in the formulae).
2. We equilibrated the number of squarings and multiplications in the formula using the technique of Longa and Miri [40] and Bernstein and Lange [9].
3. We drew a directed graph of the dependencies in the squarings and multiplications (ignoring the field additions and negations), and tried to create ordered pairs (S_i, M_i) (one squaring followed by one multiplication) that allow us to go through the graph using each operation only once.
4. Starting with the ordered pairs (S_i, M_i), we looked for the minimal numbers of field additions and negations required to include the whole formula, and tried to determine their respective position (taking special care of the first and last blocks since those tend to be the least flexible of the formula). This process led us to (S, N, A, A, M, A) blocks and a first version of the atomic block formulae, but not necessarily secured against C-safe fault attacks.
5. With used simple algebraic identities to fill all the "holes" in the formulae, for example: computing $3a$ as $2a + a$ or $2(2a) - a$, computing $4a$ as $2(2a)$ or $2a + a + a$, careful positioning of the negations (multiplications by -1). A special case of such identities will be discussed in Section 4.2.

In the following subsections, we will present the resulting atomic blocks which provide protection against both simple side-channel attacks and C-safe fault attacks.

4.2 Case 1: Left-to-right and Multi-Basis Scalar Multiplications

In this first case, we designed atomic blocks formulae in Jacobian coordinates both for "standard" left-to-right double-and-add scalar multiplications (which may use a simple binary representation, a NAF, w-NAF, etc.) as well as some multi-base scalar multiplications. The two basic operations are therefore the doubling and mixed addition (where the pre-computed points are kept in affine coordinates), which can be complemented with tripling (for double-base expansions [21]) and quintupling (for triple-base expansions [43]).

The following table compares the cost of the atomic blocks presented in [40] to those we obtained. It should be noticed that as well as giving protection against C-safe fault attacks, our atomic blocks are slightly more compact than those of Longa and Miri.

Operations	Previous work [40]	This work
Doubling	$4M + 4S + 16A + 8N$	$4M + 4S + 12A + 4N$
Mixed Addition	$6M + 6S + 24A + 12N$	$6M + 6S + 18A + 6N$
Tripling	$8M + 8S + 32A + 16N$	$8M + 8S + 24A + 8N$
Quintupling	———	$12M + 12S + 36A + 12N$

Point Doubling in Jacobian Coordinates. Let $P = (X_1 : Y_1 : Z_1)$ be a point in Jacobian coordinates on the elliptic curve E. The most efficient doubling formula (with the output also in Jacobian coordinates) requires $4M + 4S$. In terms of multiplications and squarings, there is little change from previous formulae, but we re-organize the additions and negations to fill all the operations in the atomic blocks.

$$\alpha = 3(X_1 + Z_1^2)(X_1 - Z_1^2), \qquad -\beta = (-2X_1)(2Y_1^2),$$
$$Z_3 = 2(Y_1 Z_1), \qquad X_3 = \alpha^2 - 2\beta,$$
$$Y_3 = (-\alpha)(X_3 - \beta) + 2\left[-(2Y_1^2)^2\right].$$

The resulting atomic blocks can be found in Tables 1, taking as input $R_1 \leftarrow X_1$, $R_2 \leftarrow Y_1$, and $R_3 \leftarrow Z_1$, and returning as output $X_3 \leftarrow R_1$, $Y_3 \leftarrow R_2$, and $Z_3 \leftarrow R_3$.

Mixed Addition in Jacobian-Affine Coordinates. Given the points $P = (X_1 : Y_1 : Z_1)$, in Jacobian coordinates, and $Q = (X_2, Y_2)$, in affine coordinates, both on the elliptic curve E. The most efficient formulae for the mixed addition $P + Q = (X_3 : Y_3 : Z_3)$ require $8M + 3S$. To obtain a formula practical for block atomicity, we replace two multiplications with squarings, three of which are new, to get $6M + 6S$:

$$\alpha = 2(Z_1^2 Z_1 Y_2) - 2Y_1, \qquad \beta = Z_1^2 X_2 - X_1,$$
$$2\beta^3 = (\beta^2 + \beta)^2 - (\beta^4 + \beta^2), \qquad X_3 = \alpha^2 + 2(-4X_1)\beta^2 - 4\beta^3,$$
$$Z_3 = (Z_1 + \beta)^2 - (Z_1^2 + \beta^2), \qquad Y_3 = (-\alpha)\left[(-4X_1\beta^2) + X_3\right] + (2Y_1)(-4\beta^3).$$

Table 1. Atomic block formula for Jacobian doubling

Sec	Block 1	Block 2	Block 3	Block 4
S	$R_4 \leftarrow R_3^2$ $[Z_1^2]$	$R_6 \leftarrow R_2^2$ $[Y_1^2]$	$R_4 \leftarrow R_1^2$ $[\alpha^2]$	$R_8 \leftarrow R_7^2$ $[4Y_1^4]$
N	$R_5 \leftarrow -R_4$ $[-Z_1^2]$	$R_7 \leftarrow -R_1$ $[-X_1]$	$R_5 \leftarrow -R_1$ $[-\alpha]$	$R_2 \leftarrow -R_8$ $[-4Y_1^4]$
A	$R_6 \leftarrow R_1 + R_4$ $[X_1 + Z_1^2]$	$R_1 \leftarrow R_7 + R_7$ $[-2X_1]$	$R_8 \leftarrow R_6 + R_6$ $[-2\beta]$	$R_8 \leftarrow R_1 + R_6$ $[X_3 - \beta]$
A	$R_4 \leftarrow R_1 + R_5$ $[X_1 - Z_1^2]$	$R_7 \leftarrow R_6 + R_6$ $[2Y_1^2]$	$R_1 \leftarrow R_4 + R_8$ $[X_3 = \alpha^2 - 2\beta]$	$R_4 \leftarrow R_2 + R_2$ $[-8Y_1^4]$
M	$R_5 \leftarrow R_6 R_4$ $[X_1^2 - Z_1^4]$	$R_6 \leftarrow R_1 R_7$ $[-\beta]$	$R_4 \leftarrow R_2 R_3$ $[Y_1 Z_1]$	$R_6 \leftarrow R_5 R_8$ $[-\alpha(X_3 - \beta)]$
A	$R_4 \leftarrow R_5 + R_5$ $[2(X_1^2 - Z_1^4)]$	$R_1 \leftarrow R_5 + R_4$ $[\alpha]$	$R_3 \leftarrow R_4 + R_4$ $[Z_3 = 2Y_1 Z_1]$	$R_2 \leftarrow R_6 + R_4$ $[Y_3]$

Note that the atomic blocks formulae in [40] assumed that $-Y_2^2$ had been precomputed – a reasonable assumption, but one that we do not need here, thus saving some memory and a few precomputations (for the scalar multiplication as a whole).

The resulting atomic blocks can be found in Table 2 with inputs $R_1 \leftarrow X_1$, $R_2 \leftarrow Y_1$, $R_3 \leftarrow Z_1$, $R_4 \leftarrow X_2$, and $R_5 \leftarrow Y_2$, and returning as output $X_3 \leftarrow R_1$, $Y_3 \leftarrow R_2$, and $Z_3 \leftarrow R_3$.

Point Tripling in Jacobian Coordinates. Before presenting our new atomic block tripling formula, we must make an important remark on the tripling formula of Longa and Miri [40]. Although their formula is more efficient than the one we will present here, it contains a serious weakness in terms of fault attacks. Their formulae requires $7M + 7S$, but unfortunately this formula cannot be fitted into 7 atomic blocks. When applying the formula to atomic blocks, an eighth block is required, and the first block does not use the multiplication while the eighth does not use the squaring. If two triplings are performed consecutively, the last block of the first one can be merged with the first block of the second one, so in a sequence of triplings only the first one really costs 8 blocks, all the other ones needing no more than 7 blocks. But even in a sequence of triplings, both the initial and final blocks will contain dummy operations.

Since our goal is to avoid fault attacks, we went back to the formula of Dimitrov, Imbert and Mishra [21], which had a cost of $9M + 7S$.

Let $P = (X_1 : Y_1 : Z_1)$ be a point on the elliptic curve E. To obtain our atomic blocks, we transform one multiplication into a squaring, giving $8M + 8S$:

Table 2. Atomic block formula for mixed Jacobian-affine addition

Sec	Block 1	Block 2	Block 3
S	$R_6 \leftarrow R_3^2$ $[Z_1^2]$	$R_4 \leftarrow R_1^2$ $[\beta^2]$	$R_8 \leftarrow R_4^2$ $[\beta^4]$
N	$R_1 \leftarrow -R_1$ $[-X_1]$	$R_9 \leftarrow -R_2$ $[-2Y_1]$	$R_{10} \leftarrow -R_{10}$ $[-(Z_1^2+\beta^2)]$
A	$R_2 \leftarrow R_2 + R_2$ $[2Y_1]$	$R_{10} \leftarrow R_6 + R_4$ $[Z_1^2+\beta^2]$	$R_{12} \leftarrow R_4 + R_1$ $[\beta^2+\beta]$
A	$R_8 \leftarrow R_1 + R_1$ $[-2X_1]$	$R_{11} \leftarrow R_3 + R_1$ $[Z_1+\beta]$	$R_1 \leftarrow R_8 + R_4$ $[\beta^4+\beta^2]$
M	$R_4 \leftarrow R_6 R_4$ $[Z_1^2 X_2]$	$R_6 \leftarrow R_6 R_3$ $[Z_1^3]$	$R_6 \leftarrow R_6 R_5$ $[Z_1^3 Y_2]$
A	$R_1 \leftarrow R_4 + R_1$ $[\beta]$	$R_3 \leftarrow R_8 + R_8$ $[-4X_1]$	$R_9 \leftarrow R_6 + R_9$ $[Z_1^3 Y_2 - 2Y_1]$

Sec	Block 4	Block 5	Block 6
S	$R_8 \leftarrow R_{12}^2$ $[(\beta^2+\beta)^2]$	$R_4 \leftarrow R_9^2$ $[\alpha^2]$	$R_3 \leftarrow R_{11}^2$ $[(Z_1+\beta)^2]$
N	$R_1 \leftarrow -R_1$ $[-(\beta^4+\beta^2)]$	$R_1 \leftarrow -R_1$ $[-2\beta^3]$	$R_2 \leftarrow -R_9$ $[-\alpha]$
A	$R_1 \leftarrow R_8 + R_1$ $[2\beta^3]$	$R_8 \leftarrow R_1 + R_1$ $[-4\beta^3]$	$R_7 \leftarrow R_6 + R_1$ $[-4X_1\beta^2 + X_3]$
A	$R_9 \leftarrow R_9 + R_6$ $[\alpha]$	$R_3 \leftarrow R_4 + R_3$ $[\alpha^2 - 8X_1\beta^2]$	$R_3 \leftarrow R_3 + R_{10}$ $[Z_3]$
M	$R_6 \leftarrow R_3 R_4$ $[-4X_1\beta^2]$	$R_4 \leftarrow R_2 R_8$ $[-8Y_1\beta^3]$	$R_8 \leftarrow R_2 R_7$ $[-\alpha(-4X_1\beta^2 + X_3)]$
A	$R_3 \leftarrow R_6 + R_6$ $[-8X_1\beta^2]$	$R_1 \leftarrow R_3 + R_8$ $[X_3]$	$R_2 \leftarrow R_8 + R_4$ $[Y_3]$

$$
\begin{aligned}
2\beta &= \left[(2Y_1)^2\right]^2, & \theta &= 3(X_1^2 - Z_1^4) = 3(X_1 + Z_1^2)(X_1 - Z_1^2),\\
\omega &= 3X_1(4Y_1^2) - \theta^2, & 4X_1\omega^2 &= (2X_1 + \omega^2)^2 - (2X_1)^2 - (\omega^2)^2,\\
2\alpha &= 2\theta\omega, & \rho &= (2\alpha - 2\beta)(4\beta - 2\alpha),\\
Z_3 &= (2Z_1)\omega, & Y_3 &= 2(-4Y_1)\left[\omega^2\omega - \rho\right],\\
X_3 &= 4(4Y_1^2)(2\beta - 2\alpha) + (4X_1\omega^2).
\end{aligned}
$$

The resulting atomic blocks can be found in Table 3, taking as input $R_1 \leftarrow X_1$, $R_2 \leftarrow Y_1$, and $R_3 \leftarrow Z_1$, and returning as output $X_3 \leftarrow R_1$, $Y_3 \leftarrow R_2$, and $Z_3 \leftarrow R_3$.

Point Quintupling in Jacobian Coordinates. Let $P = (X_1 : Y_1 : Z_1)$ be a point on the elliptic curve E. The formula of Mishra and Dimitrov [43] requires $15M + 18S$. This formula was improved by Longa and Miri [39] to a balanced $11M+11S$ but these field operations cannot be fitted into a sequences of (S_i, M_i) pairs. To obtain atomic blocks, we re-balance the formula to $12M+12S$

Table 3. Atomic block formula for Jacobian tripling

Sec	Block 1	Block 2	Block 3	Block 4
S	$R_4 \leftarrow R_3^2$ $[Z_1^2]$	$R_5 \leftarrow R_5^2$ $[(2Y_1)^2]$	$R_7 \leftarrow R_4^2$ $[\theta^2]$	$R_7 \leftarrow R_5^2$ $[2\beta]$
N	$R_5 \leftarrow -R_4$ $[-Z_1^2]$	$R_2 \leftarrow -R_2$ $[-Y_1]$	$R_7 \leftarrow -R_7$ $[-\theta^2]$	$R_4 \leftarrow -R_4$ $[-2\alpha]$
A	$R_4 \leftarrow R_1 + R_4$ $[X_1 + Z_1^2]$	$R_6 \leftarrow R_4 + R_4$ $[2(X_1^2 - Z_1^4]$	$R_6 \leftarrow R_9 + R_6$ $[12X_1Y_1^2]$	$R_4 \leftarrow R_7 + R_4$ $[2\beta - 2\alpha]$
A	$R_5 \leftarrow R_1 + R_5$ $[X_1 - Z_1^2]$	$R_4 \leftarrow R_6 + R_4$ $[\theta]$	$R_6 \leftarrow R_6 + R_7$ $[\omega]$	$R_7 \leftarrow R_4 + R_7$ $[4\beta - 2\alpha]$
M	$R_4 \leftarrow R_4R_5$ $[X_1^2 - Z_1^4]$	$R_6 \leftarrow R_1R_5$ $[4X_1Y_1^2]$	$R_7 \leftarrow R_6R_4$ $[\omega\theta]$	$R_5 \leftarrow R_5R_4$ $[4Y_1^2(2\beta - 2\alpha)]$
A	$R_5 \leftarrow R_2 + R_2$ $[2Y_1]$	$R_9 \leftarrow R_6 + R_6$ $[8X_1Y_1^2]$	$R_4 \leftarrow R_7 + R_7$ $[2\alpha]$	$R_5 \leftarrow R_5 + R_5$ $[8Y_1^2(2\beta - 2\alpha)]$

Sec	Block 5	Block 6	Block 7	Block 8
S	$R_8 \leftarrow R_6^2$ $[\omega^2]$	$R_{10} \leftarrow R_8^2$ $[\omega^4]$	$R_1 \leftarrow R_1^2$ $[4X_1^2]$	$R_7 \leftarrow R_3^2$ $[(2X_1 + \omega^2)^2]$
N	$R_4 \leftarrow -R_4$ $[2\alpha - 2\beta]$	$R_{10} \leftarrow -R_{10}$ $[-\omega^4]$	$R_4 \leftarrow -R_4$ $[-\rho]$	$R_8 \leftarrow -R_1$ $[-4X_1^2]$
A	$R_5 \leftarrow R_5 + R_5$ $[16Y_1^2(2\beta - 2\alpha)]$	$R_1 \leftarrow R_1 + R_1$ $[2X_1]$	$R_8 \leftarrow R_8 + R_4$ $[\omega^3 - \rho]$	$R_1 \leftarrow R_8 + R_{10}$ $[-4X_1^2 - \omega^4]$
A	$R_9 \leftarrow R_3 + R_3$ $[2Z_1]$	$R_3 \leftarrow R_1 + R_8$ $[2X_1 + \omega^2]$	$R_7 \leftarrow R_7 + R_2$ $[-4Y_1]$	$R_4 \leftarrow R_7 + R_1$ $[4X_1\omega^2]$
M	$R_4 \leftarrow R_4R_7$ $[\rho]$	$R_8 \leftarrow R_8R_6$ $[\omega^3]$	$R_4 \leftarrow R_7R_8$ $[-4Y_1R_8]$	$R_3 \leftarrow R_9R_6$ $[Z_3]$
A	$R_7 \leftarrow R_2 + R_2$ $[-2Y_1]$	$R_7 \leftarrow R_7 + R_2$ $[-3Y_1]$	$R_2 \leftarrow R_4 + R_4$ $[Y_3]$	$R_1 \leftarrow R_5 + R_4$ $[X_3]$

(slightly less efficient, but possible to fit into 12 atomic blocks). To fill all the spaces for field additions, we take advantage of the following substitution $-E^2 = M^2 + 2ME - (M + E)^2$, where M^2 and ME are required in other parts of the computations, so the new computation will still give $-E^2$ with a different squaring, but using more additions. We obtain the following formula:

$$
\begin{aligned}
2T &= \left[(2Y_1)^2\right]^2, & M &= 3(X_1 + Z_1^2)(X_1 - Z_1^2),\\
E &= 3X_1(4Y_1^2) + (-M^2), & 2L &= 2ME - 2T,\\
-E^2 &= M^2 + 2ME - (M + E)^2, & 16L^4 &= \left[(2L)^2\right]^2,\\
2U &= (2Y_1 + 2L)^2 - (2Y_1)^2 - (2L)^2, & V &= (2T)(2L) + E(-E^2),\\
N &= V - (2L)^2, & 2W &= 2EN = (E + N)^2 - E^2 - N^2,\\
S &= V^2 + 16L^4 - 3V(4L^2), & Z_5 &= (Z_1 + V)^2 - Z_1^2 - V^2,\\
Y_5 &= 2(2Y_1)\Big[2(-E^3)S + 2\big[-(2T)(2L)(16L^4)\big]\Big], & X_5 &= 4X_1V^2 - (2Y_1)(2U)(2W).
\end{aligned}
$$

Table 4. Atomic block formula for Jacobian quintupling (operations and registers)

Sec	Block 1	Block 2	Block 3	Block 4
S	$R_4 \leftarrow R_3^2$	$R_6 \leftarrow R_2^2$	$R_{10} \leftarrow R_8^2$	$R_4 \leftarrow R_6^2$
N	$R_5 \leftarrow -R_4$	$R_7 \leftarrow -R_6$	$R_{11} \leftarrow -R_{10}$	$R_6 \leftarrow -R_4$
A	$R_6 \leftarrow R_1 + R_4$	$R_8 \leftarrow R_4 + R_4$	$R_9 \leftarrow R_9 + R_4$	$R_{14} \leftarrow R_{11} + R_6$
A	$R_4 \leftarrow R_1 + R_5$	$R_8 \leftarrow R_4 + R_8$	$R_9 \leftarrow R_9 + R_{11}$	$R_8 \leftarrow R_9 + R_8$
M	$R_4 \leftarrow R_6 R_4$	$R_4 \leftarrow R_1 R_6$	$R_4 \leftarrow R_8 R_9$	$R_6 \leftarrow R_4 R_{14}$
A	$R_2 \leftarrow R_2 + R_2$	$R_9 \leftarrow R_4 + R_4$	$R_{11} \leftarrow R_4 + R_4$	$R_4 \leftarrow R_{11} + R_{10}$

Sec	Block 5	Block 6	Block 7	Block 8
S	$R_{10} \leftarrow R_8^2$	$R_{12} \leftarrow R_4^2$	$R_{14} \leftarrow R_{14}^2$	$R_{11} \leftarrow R_{11}^2$
N	$R_{11} \leftarrow -R_{10}$	$R_{15} \leftarrow -R_{12}$	$R_3 \leftarrow -R_{14}$	$R_{15} \leftarrow -R_{15}$
A	$R_8 \leftarrow R_4 + R_{11}$	$R_5 \leftarrow R_{15} + R_5$	$R_7 \leftarrow R_3 + R_7$	$R_{11} \leftarrow R_{11} + R_7$
A	$R_{11} \leftarrow R_2 + R_{14}$	$R_{13} \leftarrow R_3 + R_4$	$R_3 \leftarrow R_4 + R_3$	$R_7 \leftarrow R_9 + R_3$
M	$R_{10} \leftarrow R_9 R_8$	$R_3 \leftarrow R_1 R_{15}$	$R_4 \leftarrow R_4 R_{14}$	$R_{11} \leftarrow R_2 R_{11}$
A	$R_4 \leftarrow R_6 + R_{10}$	$R_1 \leftarrow R_3 + R_3$	$R_{15} \leftarrow R_4 + R_4$	$R_9 \leftarrow R_{15} + R_{15}$

Sec	Block 9	Block 10	Block 11	Block 12
S	$R_{15} \leftarrow R_{14}^2$	$R_4 \leftarrow R_7^2$	$R_{10} \leftarrow R_3^2$	$R_7 \leftarrow R_{13}^2$
N	$R_1 \leftarrow -R_1$	$R_{14} \leftarrow -R_{12}$	$R_1 \leftarrow -R_{10}$	$R_6 \leftarrow -R_{11}$
A	$R_{14} \leftarrow R_{12} + R_{15}$	$R_{15} \leftarrow R_{14} + R_{14}$	$R_9 \leftarrow R_8 + R_1$	$R_3 \leftarrow R_7 + R_5$
A	$R_4 \leftarrow R_9 + R_4$	$R_{14} \leftarrow R_1 + R_1$	$R_7 \leftarrow R_{12} + R_{15}$	$R_{12} \leftarrow R_4 + R_9$
M	$R_{12} \leftarrow R_6 R_{15}$	$R_9 \leftarrow R_{10} R_6$	$R_7 \leftarrow R_2 R_7$	$R_{12} \leftarrow R_6 R_{12}$
A	$R_6 \leftarrow R_{14} + R_4$	$R_{12} \leftarrow R_9 + R_9$	$R_2 \leftarrow R_7 + R_7$	$R_1 \leftarrow R_{14} + R_{12}$

The resulting atomic blocks can be found in Table 4 (for formating reasons, the equivalences for each register are given in Table 5). The formula takes as input $R_1 \leftarrow X_1$, $R_2 \leftarrow Y_1$, and $R_3 \leftarrow Z_1$, and returns as output $X_3 \leftarrow R_1$, $Y_3 \leftarrow R_2$, and $Z_3 \leftarrow R_3$.

4.3 Case 2: Right-to-left Scalar Multiplication

We now consider atomic blocks formulae designed for right-to-left scalar multiplications based on Yao's algorithm [51,5,32]. These scalar multiplications have the advantage of avoiding the Doubling attack of Fouque and Valette [23]. The first atomic formulae designed for this context are due to Giraud and Verneuil [26]. The two basic operations for these scalar multiplications are the doubling and addition (with inputs in Jacobian coordinates). In optimal (side-channel unsafe) implementations, Modified Jacobian coordinates offer a performance gain on Jacobian coordinates, but this gain is lost with atomic blocks formulae, as the number of blocks for doublings is the same. All the operations can then be performed in Jacobian coordinates. We give the Modified Jacobian doubling in Appendix A but recommend using the doubling formula in Table 1.

The following table compares the cost of the atomic blocks presented in [26] with those we obtained. Note that the structure of the atomic blocks is quite different, but the main difference here comes from the added security against C-safe attacks.

Table 5. Atomic block formula for Jacobian quintupling (equivalences)

Sec	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6
S	$[Z_1^2]$	$[4Y_1^2]$	$[M^2]$	$[2T]$	$[(E+M)^2]$	$[V^2]$
N	$[-Z_1^2]$	$[-4Y_1^2]$	$[-M^2]$	$[-2T]$	$[-(E+M)^2]$	$[-V^2]$
A	$[X_1+Z_1^2]$	$[2(X_1^2-Z_1^4)]$	$[3X_1(2Y_1)^2]$	$[2L]$	$[-E^2]$	$[-V^2-Z_1^2]$
A	$[X_1-Z_1^2]$	$[M]$	$[E]$	$[E+M]$	$[2Y_1+2L]$	$[Z_1+V]$
M	$[X_1^2-Z_1^4]$	$[4X_1Y_1^2]$	$[ME]$	$[4TL]$	$[-E^3]$	$[-X_1V^2]$
A	$[2Y_1]$	$[8X_1Y_1^2]$	$[2ME]$	$[2ME+M^2]$	$[V]$	$[-2X_1V^2]$

Sec	Block 7	Block 8	Block 9	Block 10	Block 11	Block 12
S	$[4L^2]$	$[4(Y_1+L)^2]$	$[16L^4]$	$[(E+N)^2]$	$[N^2]$	$[(Z_1+V)^2]$
N	$[-4L^2]$	$[-8VL^2]$	$[2X_1V^2]$	$[-64TL^5]$	$[-N^2]$	$[-4Y_1U]$
A	$[-4L^2-4Y_1^2]$	$[2U]$	$[V^2+16L^4]$	$[-128TL^5]$	$[-E^2-N^2]$	$[Z_5]$
A	$[N]$	$[E+N]$	$[-12VL^2]$	$[4X_1V^2]$	$[Y_5/4Y_1]$	$[2W]$
M	$[4VL^2]$	$[4Y_1U]$	$[64TL^5]$	$[-E^3S]$	$[Y_5/2]$	$[-8Y_1UW]$
A	$[8VL^2]$	$[-16VL^2]$	$[S]$	$[-2E^3S]$	$[Y_5]$	$[X_5]$

Operations	Previous work [26]	This work
(Modified) Jacobian Doubling	$6M+2S+10A+4N$	$4M+4S+12A+4N$
General Jacobian Addition	$12M+4S+20A+6N$	$9M+9S+27A+9N$

Table 6. Atomic block formula for general Jacobian addition (operations and registers)

Sec	Block 1	Block 2	Block 3
S	$R_7 \leftarrow R_3^2$	$R_3 \leftarrow R_6^2$	$R_{13} \leftarrow R_{11}^2$
N	$R_8 \leftarrow -R_7$	$R_{12} \leftarrow -R_3$	$R_2 \leftarrow -R_2$
A	$R_9 \leftarrow R_7+R_3$	$R_6 \leftarrow R_3+R_6$	$R_{14} \leftarrow R_{13}+R_{13}$
A	$R_{10} \leftarrow R_2+R_2$	$R_2 \leftarrow R_{10}+R_2$	$R_{15} \leftarrow R_{11}+R_{11}$
M	$R_{11} \leftarrow R_4R_7$	$R_1 \leftarrow R_1R_{12}$	$R_{14} \leftarrow R_{14}R_{15}$
A	$R_4 \leftarrow R_3+R_6$	$R_{11} \leftarrow R_{11}+R_1$	$R_{15} \leftarrow R_8+R_{12}$

Sec	Block 4	Block 5	Block 6
S	$R_3 \leftarrow R_3^2$	$R_6 \leftarrow R_6^2$	$R_2 \leftarrow R_7^2$
N	$R_3 \leftarrow -R_3$	$R_{11} \leftarrow -R_{11}$	$R_2 \leftarrow -R_2$
A	$R_3 \leftarrow R_3+R_{12}$	$R_{16} \leftarrow R_6+R_3$	$R_{16} \leftarrow R_2+R_8$
A	$R_2 \leftarrow R_2+R_{10}$	$R_{12} \leftarrow R_{10}+R_{10}$	$R_8 \leftarrow R_6+R_1$
M	$R_1 \leftarrow R_1R_{13}$	$R_3 \leftarrow R_2R_{16}$	$R_7 \leftarrow R_3R_{14}$
A	$R_{10} \leftarrow R_1+R_1$	$R_6 \leftarrow R_{12}+R_{10}$	$R_2 \leftarrow R_8+R_1$

Sec	Block 7	Block 8	Block 9
S	$R_8 \leftarrow R_9^2$	$R_{16} \leftarrow R_4^2$	$R_{16} \leftarrow R_9^2$
N	$R_1 \leftarrow -R_{14}$	$R_5 \leftarrow -R_9$	$R_3 \leftarrow -R_{13}$
A	$R_{14} \leftarrow R_8+R_{16}$	$R_{14} \leftarrow R_2+R_{12}$	$R_1 \leftarrow R_{16}+R_{10}$
A	$R_{10} \leftarrow R_1+R_2$	$R_4 \leftarrow R_{16}+R_{15}$	$R_{12} \leftarrow R_{16}+R_4$
M	$R_6 \leftarrow R_5R_{14}$	$R_{13} \leftarrow R_{11}R_4$	$R_{14} \leftarrow R_5R_{12}$
A	$R_9 \leftarrow R_6+R_3$	$R_4 \leftarrow R_1+R_{14}$	$R_2 \leftarrow R_{14}+R_7$

General Addition in Jacobian Coordinates. Given the points $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$ in Jacobian coordinates, both on the elliptic curve E. The most efficient formulae for the addition $P + Q = (X_3 : Y_3 : Z_3)$ require $11M + 5S$.

To obtain a formula practical for block atomicity, we replace 3 multiplications with 5 squarings, to get $9M + 9S$:

$$
\begin{aligned}
-A &= X_1(-Z_2^2), & B &= X_2 Z_1^2, \\
E &= B - A, & -2C &= -Y_1\left[(Z_2^2 + Z_2)^2 - (Z_2^2)^2 - Z_2^2\right], \\
2D &= Y_2\left[(Z_1^2 + Z_1)^2 - (Z_1^2)^2 - Z_1^2\right], & 2F &= 2D - 2C, \\
X_3 &= (2F)^2 - (2E^2)(2E) + 8(-AE^2), & Z_3 &= E\left[(Z_1 + Z_2)^2 - Z_1^2 - Z_2^2\right], \\
Y_3 &= -2F\left[(2F)^2 - (2E^2)(2E) + 12(-AE^2)\right] - 2C(2E^2)(2E).
\end{aligned}
$$

The resulting atomic blocks can be found in Table 6 (for formating reasons, the equivalences for each register are given in Table 7). The formula takes as input $R_1 \leftarrow X_1$, $R_2 \leftarrow Y_1$, $R_3 \leftarrow Z_1$, $R_4 \leftarrow X_2$, $R_5 \leftarrow Y_2$, $R_6 \leftarrow Z_2$, and returns as output $X_3 \leftarrow R_1$, $Y_3 \leftarrow R_2$, and $Z_3 \leftarrow R_3$.

Table 7. Atomic block formula for general Jacobian addition (equivalences)

Sec	Block 1	Block 2	Block 3
S	$[Z_1^2]$	$[Z_2^2]$	$[E^2]$
N	$[-Z_1^2]$	$[-Z_2^2]$	$[-3Y_1]$
A	$[Z_1^2 + Z_1]$	$[Z_2^2 + Z_2]$	$[2E^2]$
A	$[2Y_1]$	$[3Y_1]$	$[2E]$
M	$[B]$	$[-A]$	$[4E^3]$
A	$[Z_1 + Z_2]$	$[E]$	$[-Z_1^2 - Z_2^2]$

Sec	Block 4	Block 5	Block 6
S	$[Z_2^4]$	$[(Z_2^2 + Z_2)^2]$	$[Z_1^4]$
N	$[-Z_2^4]$	$[-E]$	$[-Z_1^4]$
A	$[-Z_2^4 - Z_2^2]$	$[(Z_2^2 + Z_2)^2 - Z_2^4 - Z_2^2]$	$[-Z_1^4 - Z_1^2]$
A	$[-Y_1]$	$[-4AE^2]$	$[-7AE^2]$
M	$[-AE^2]$	$[-2C]$	$[-8CE^3]$
A	$[-2AE^2]$	$[-6AE^2]$	$[-8AE^2]$

	Block 7	Block 8	Block 9
S	$[(Z_1^2 + Z_1)^2]$	$[(Z_1 + Z_2)^2]$	$[4F^2]$
N	$[-4E^3]$	$[-2F]$	$[Z_3]$
A	$[(Z_1^2 + Z_1)^2 - Z_1^4 - Z_1^2]$	$[-12AE^2]$	$[X_3]$
A	$[-4E^3 - 8AE^2]$	$[(Z_1 + Z_2)^2 - Z_1^2 - Z_2^2]$	$[X_3 - 4AE^2]$
M	$[2D]$	$[-Z_3]$	$[-2F(X_3 - 4AE^2)]$
A	$[2F = 2D - 2C]$	$[-4E^3 - 12AE^2]$	$[Y_3]$

4.4 Case 3: Unified Formulae

In [15], Chen *et al.* recommend using atomic blocks formulae which takes the same number of blocks for both the group additions and doublings. Although a careful handling of the time interval between blocks is preferable (for efficiency reasons), our techniques can be used to obtain a doubling formula that is unified with the mixed addition. The details of the formula can be found in Appendix A.

5 Conclusion

The aim of our paper was to give an improvement on previous atomic block formulae [40,26,15], securing them against C-safe fault attacks [52,4]. This was done by adjusting the formulae so that all the field operations of every atomic block have an impact on the final result (i.e. there are no dummy operations), while avoiding the repetition of any single operation. A careful implementation of our atomic blocks formulae (which should include a careful management of delays between each atomic blocks) will then secure the scalar multiplication against both SSCA and C-safe fault attacks.

As well as offering increased security, our atomic blocks are slightly more compact than those of Longa and Miri [40]. Assuming experimental average ratios to multiplications of $S/M \approx 0.8$, $A/M \approx 0.2$ and $N/M \approx 0.1$ [26], each of our atomic block takes $\approx 2.5M$ compared to $\approx 2.8M$ for those of Longa and Miri. This representing savings of close to 10% on the whole scalar multiplication, both in double-and-add algorithms and multi-basis (2, 3, and 5) scalar multiplications. For right-to-left scalar multiplications, the comparison with the atomic blocks of Giraud and Verneuil [26] is more difficult because of the difference in the block structures. For the group doublings, the total costs are very close, but for the general Jacobian addition our block formula costs $\approx 22.5M$ compared to $\approx 20M$ for Giraud and Verneuil. This means an increased cost of close to 10% to additions (and less than that over the whole scalar multiplication) for the added resistance to C-safe fault attacks, a very reasonable price in most contexts.

References

1. Amiel, F., Feix, B., Tunstall, M., Whelan, C., Marnane, W.P.: Distinguishing Multiplications from Squaring Operations. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 346–360. Springer, Heidelberg (2009)
2. Avanzi, R., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Champan & Hall/CRC Press (2005)
3. Avanzi, R.: Aspects of Hyperelliptic Curves over Large Prime Fields in Software Implementations. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 148–162. Springer, Heidelberg (2004)
4. Avanzi, R.: Side Channel Attacks on Implementations of Curve-Based Cryptographic Primitives. Cryptology ePrint Archive Report 2005/017, http://eprint.iacr.org/

5. Avanzi, R.: Delaying and Merging Operations in Scalar Multiplication: Applications to Curve-Based Cryptosystems. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 203–219. Springer, Heidelberg (2007)
6. Bernstein, D.J., Lange, T.: Inverted Edwards Coordinates. In: Boztaş, S., Lu, H.-F. (eds.) AAECC 2007. LNCS, vol. 4851, pp. 20–27. Springer, Heidelberg (2007)
7. Bernstein, D.J., Lange, T.: Explicit Formulae Database, http://www.hyperelliptic.org/EFD/
8. Bernstein, D.J.: Curve25519: New Diffie-Hellman Speed Records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (2006)
9. Bernstein, D.J., Lange, T.: Faster Addition and Doubling on Elliptic Curves. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 29–50. Springer, Heidelberg (2007)
10. Billet, O., Joye, M.: The Jacobi Model of an Elliptic Curve and Side-Channel Analysis. In: Fossorier, M.P.C., Høholdt, T., Poli, A. (eds.) AAECC 2003. LNCS, vol. 2643, pp. 34–42. Springer, Heidelberg (2003)
11. Brier, E., Joye, M.: Weierstrass Elliptic Curve and Side-Channel Attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002)
12. Brier, E., Dechene, I., Joye, M.: Unified Point Addition Formulae for Elliptic Curve Cryptosystems. In: Embedded Cryptographic Hardware: Methodologies and Architectures, pp. 247–256. Nova Science Publishers (2004)
13. Brier, E., Joye, M.: Fast Point Multiplication on Elliptic Curves Through Isogenies. In: Fossorier, M.P.C., Høholdt, T., Poli, A. (eds.) AAECC 2003. LNCS, vol. 2643, pp. 43–50. Springer, Heidelberg (2003)
14. Brown, M., Hankerson, D., López, J., Menezes, A.: Software Implementation of the NIST Elliptic Curves over Prime Fields. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 250–265. Springer, Heidelberg (2001)
15. Chen, T., Li, H., Wu, K., Yu, F.: Countermeasure of ECC against Side-channel Attacks: Balanced Point Addition and Point Doubling Operation Procedure. In: IEEE, Asia-Pacific Conference on Information Processing, vol. 2, pp. 465–469. IEEE Conference Publications (2009)
16. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. IEEE Trans. Computers 53(6), 760–768 (2004)
17. Chong Hee, K., Quisquater, J.J.: Faults, Injection Methods, and Fault Attacks. IEEE Design & Test of Computers 24(6), 544–545 (2007)
18. Chudnovsky, D.V., Chudnovsky, G.V.: Sequences of Numbers Generated by Addtion in Formal Groups and New Primality and Factorization Test. Advances in Applied Mathematics 7, 385–434 (1986)
19. Cohen, H., Ono, T., Miyaji, A.: Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 51–65. Springer, Heidelberg (1998)
20. Coron, J.: Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
21. Dimitrov, V., Imbert, L., Mishra, P.K.: Efficient and Secure Elliptic Curve Point Multiplication Using Double-Base Chains. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 59–78. Springer, Heidelberg (2005)
22. Edwards, H.M.: A Normal Form for Elliptic Curves. Bull. Am. Math. Soc., New Ser. 44(3), 393–422 (2007)

23. Fouque, P.-A., Valette, F.: The Doubling Attack: Why Upwards Is Better than Downwards. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 269–280. Springer, Heidelberg (2003)
24. Gandolfi, K., Mourtel, C., Olivier, F.: Electronic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
25. Gebotys, C.H., Gebotys, R.J.: Secure Elliptic Curve Implementations: An Analysis of Resistance to Power-Attacks in a DSP Processor. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 114–128. Springer, Heidelberg (2003)
26. Giraud, C., Verneuil, V.: Atomicity Improvement for Elliptic Curve Scalar Multiplication. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 80–101. Springer, Heidelberg (2010)
27. Großschaldl, J., Avanzi, R., Savas, E., Tillich, S.: Energy-Efficient Software Implementation of Long Interger Modular Arithmetic. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 75–90. Springer, Heidelberg (2005)
28. Goundar, R.R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar Multiplication on Weierstrass Elliptic Curves from Co-Z Arithmetic. Journal of Cryptographic Engineering 1(2), 161–176 (2011)
29. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C.: Comparing Elliptic Curve on RSA on 8-bit CPUs. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 119–132. Springer, Heidelberg (2004)
30. Hanley, N., Tunstall, M., Marmane, W.P.: Using Templates to Distinguishing Multiplications from Squaring Operations. International Journal Information Security 10(4), 255–266 (2011)
31. Joye, M.: Highly Regular Right-to Left Algorithms for Scalar Multiplication. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 135–147. Springer, Heidelberg (2007)
32. Joye, M.: Fast Point Multiplication on Elliptic Curves Without Precomputation. In: von zur Gathen, J., Imaña, J.L., Koç, Ç.K. (eds.) WAIFI 2008. LNCS, vol. 5130, pp. 36–46. Springer, Heidelberg (2008)
33. Joye, M., Tibouchi, M., Vergnaud, D.: Huff's Model for Elliptic Curves. In: Hanrot, G., Morain, F., Thomé, E. (eds.) ANTS-IX. LNCS, vol. 6197, pp. 234–250. Springer, Heidelberg (2010)
34. Joye, M., Yen, S.-M.: The Montgomery Powering Ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
35. Koblitz, N.: Elliptic Curve Cryptosystems. Mathematics of Computation 48, 203–209 (1987)
36. Kocher, P.: Timing Attacks on Implementation of Diffie-Hellman RSA, DSS and other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
37. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
38. Liardet, P.Y., Smart, N.P.: Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 401–411. Springer, Heidelberg (2001)
39. Longa, P., Miri, A.: New Multibase Non-Adjacent Form Scalar Multiplication and its Application to Elliptic Curve Cryptosystems (extended version). Crytology ePrint Archive, Report 2008/052, http://eprint.iacr.org/

40. Longa, P., Miri, A.: Fast and Flexible Elliptic Curves Point Arithmetic over Prime Fields. IEEE Trans. on Computers 57(3), 289–302 (2008)
41. López, J., Dahab, R.: Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316–327. Springer, Heidelberg (1999)
42. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
43. Mishra, P.K., Dimitrov, V.: Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication Using Multibase Number Representation. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 390–406. Springer, Heidelberg (2007)
44. Möller, B.: Securing Elliptic Curve Point Multiplication against Side-channel Attacks. In: Davida, G.I., Frankel, Y. (eds.) ISC 2001. LNCS, vol. 2200, pp. 324–334. Springer, Heidelberg (2001)
45. Montgomery, P.: Speeding the Pollard and Elliptic Curve methods of Factorization. Mathematics of Computation 48(177), 243–264 (1987)
46. 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)
47. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
48. Smart, N.P.: The Hessian Form of an Elliptic Curve. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 118–125. Springer, Heidelberg (2001)
49. Thériault, N.: SPA Resistant Left-to-Right Integer Recodings. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 345–358. Springer, Heidelberg (2006)
50. Tunstall, M., Joye, M.: Coordinate Blinding over Large Prime Fields. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 443–445. Springer, Heidelberg (2010)
51. Yao, A.C.: On the Evaluation of Powers. SIAM Journal on Computing 5, 100–103 (1976)
52. Yen, S.-M., Joye, M.: Checking Before Output not be Enough Against Fault-based Cryptanalysis. IEEE Trans. on Computers 49(9), 967–970 (2000)
53. Yen, S.-M., Kim, S., Lim, S., Moon, S.: A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 414–427. Springer, Heidelberg (2002)

A Supplementary Operations

Point Doubling in Modified Jacobian Coordinates. The Modified Jacobian coordinates were introduced by Cohen *et al.* [19]. They are based on Jacobian coordinates $(X : Y : Z)$, but with an extra coordinate $W = aZ^4$. This allows a doubling operation with $3M + 5S$, at a lower cost than for the shorter Jacobian coordinates. Obtaining an equilibrated formula for this operation will naturally require at least $4M + 4S$.

Let $P = (X_1 : Y_1 : Z_1 : W_1)$ be a point in Modified Jacobian coordinates. One can be compute $2P = (X_2 : Y_2 : Z_2 : W_2)$ by the following formula with

complexity $4M + 4S + 12A + 3N$:

$$\begin{aligned} A &= 3X_1^2 + W_1, & B &= 2Y_1^2, \\ -C &= 2B(-X_1), & -D &= 2(-B^2), \\ X_2 &= A^2 - 2C, & Z_2 &= (2Y_1)Z_1, \\ Y_2 &= -A(X_2 - C) - D, & W_2 &= (-2D)(-W_1). \end{aligned}$$

The resulting atomic blocks can be found in Table 8 with inputs $R_1 \leftarrow X_1$, $R_2 \leftarrow Y_1$, $R_3 \leftarrow Z_1$, and $R_4 \leftarrow W_1$, and returning as output $X_2 \leftarrow R_1$, $Y_2 \leftarrow R_2$, $Z_2 \leftarrow R_3$, and $W_2 \leftarrow R_5$.

Table 8. Atomic block formula for Modified Jacobian Doubling

Sec	Block 1	Block 2	Block 3	Block 4
S	$R_5 \leftarrow R_1^2$ $[X_1^2]$	$R_5 \leftarrow R_2^2$ $[Y_1^2]$	$R_5 \leftarrow R_5^2$ $[B^2]$	$R_4 \leftarrow R_1^2$ $[A^2]$
N	$R_6 \leftarrow -R_1$ $[-X_1]$	$R_7 \leftarrow -R_4$ $[-W_1]$	$R_5 \leftarrow -R_5$ $[-B^2]$	$R_7 \leftarrow -R_1$ $[-A]$
A	$R_7 \leftarrow R_2 + R_2$ $[2Y_1]$	$R_5 \leftarrow R_5 + R_5$ $[B]$	$R_8 \leftarrow R_5 + R_5$ $[-D]$	$R_1 \leftarrow R_4 + R_6$ $[X_2]$
A	$R_1 \leftarrow R_5 + R_5$ $[2X_1^2]$	$R_2 \leftarrow R_5 + R_5$ $[2B]$	$R_5 \leftarrow R_8 + R_8$ $[-2D]$	$R_4 \leftarrow R_1 + R_2$ $[X_2 - C]$
M	$R_3 \leftarrow R_7 R_3$ $[Z_2]$	$R_2 \leftarrow R_2 R_6$ $[-C]$	$R_5 \leftarrow R_5 R_7$ $[W_2]$	$R_7 \leftarrow R_7 R_4$ $[-A(X_2 - C)]$
A	$R_1 \leftarrow R_1 + R_5$ $[3X_1^2]$	$R_6 \leftarrow R_2 + R_2$ $[-2C]$	$R_1 \leftarrow R_1 + R_4$ $[A]$	$R_2 \leftarrow R_7 + R_8$ $[Y_2]$

Point Doubling in Jacobian Coordinates Balanced with Mixed Addition. In [15], Chen *et al.* presented a new experimental attack on smart cards using the atomic blocks proposed by Chevallier-Mames *et al.* [16] with (M, A, N, A) structure. This experimental attack is applied because the implementation of Chen *et al.* does not avoid breaks between doubling and Jacobian addition. As a countermeasure to their attack, they presented a new atomic blocks formula for doubling which is equilibrated with the addition formula.

The ideal solution would be to refine the implementation so that the time between each block is always the same.The alternative, as Chen *et al.* proposed, is to rework the doubling formula to use the same number of blocks as the addition. Our techniques to obtain full atomic blocks can be used after "balancing" the doubling formula to $6M + 6S$.

Operations	Previous work [15]	This work
Doubling	$16M + 32A + 16N$	$6M + 6S + 18A + 6N$

We now present a version of the Jacobian doubling formula which uses 6 atomic blocks. Let $P = (X_1 : Y_1 : Z_1)$ be a point in Jacobian coordinates. One can be compute $2P = (X_3 : Y_3 : Z_3)$ by the following formula with complexity

Table 9. Atomic block formula for doubling balanced with mixed addition

Sec	Block 1	Block 2	Block 3
S	$R_4 \leftarrow R_3^2$ $[Z_1^2]$	$R_1 \leftarrow R_2^2$ $[Y_1^2]$	$R_2 \leftarrow R_6^2$ $[4X_1^2]$
N	$R_5 \leftarrow -R_4$ $[-Z_1^2]$	$R_6 \leftarrow -R_4$ $[-2X_1]$	$R_7 \leftarrow -R_5$ $[-\alpha]$
A	$R_4 \leftarrow R_1 + R_4$ $[X_1 + Z_1^2]$	$R_7 \leftarrow R_5 + R_5$ $[2(X_1^2 - Z_1^4)]$	$R_8 \leftarrow R_1 + R_1$ $[2Y_1^2]$
A	$R_5 \leftarrow R_1 + R_5$ $[X_1 - Z_1^2]$	$R_5 \leftarrow R_7 + R_5$ $[\alpha]$	$R_4 \leftarrow R_8 + R_4$ $[2Y_1^2 + 2X_1]$
M	$R_5 \leftarrow R_5 R_4$ $[X_1^2 - Z_1^4]$	$R_7 \leftarrow R_2 R_3$ $[Y_1 Z_1]$	$R_9 \leftarrow R_6 R_4$ $[-2X_1(2Y_1^2 + 2X_1)]$
A	$R_4 \leftarrow R_1 + R_1$ $[2X_1]$	$R_3 \leftarrow R_7 + R_7$ $[Z_3]$	$R_6 \leftarrow R_9 + R_2$ $[-\beta]$

Sec	Block 4	Block 5	Block 6
S	$R_9 \leftarrow R_6^2$ $[\beta^2]$	$R_9 \leftarrow R_5^2$ $[\alpha^2]$	$R_9 \leftarrow R_8^2$ $[4Y_1^4]$
N	$R_2 \leftarrow -R_6$ $[\beta]$	$R_9 \leftarrow -R_9$ $[-\alpha^2]$	$R_5 \leftarrow -R_9$ $[-4Y_1^4]$
A	$R_4 \leftarrow R_5 + R_2$ $[\alpha + \beta]$	$R_{10} \leftarrow R_6 + R_6$ $[-2\beta]$	$R_6 \leftarrow R_5 + R_5$ $[-8Y_1^4]$
A	$R_2 \leftarrow R_5 + R_6$ $[\alpha - \beta]$	$R_1 \leftarrow R_2 + R_{10}$ $[X_3]$	$R_5 \leftarrow R_{10} + R_6$ $[\alpha\beta - 8Y_1^4]$
M	$R_{10} \leftarrow R_4 R_2$ $[\alpha^2 - \beta^2]$	$R_6 \leftarrow R_5 R_4$ $[\alpha(\alpha + \beta)]$	$R_6 \leftarrow R_7 R_1$ $[-\alpha X_3]$
A	$R_2 \leftarrow R_{10} + R_9$ $[(\alpha^2 - \beta^2) + \beta^2]$	$R_{10} \leftarrow R_6 + R_9$ $[\alpha\beta]$	$R_2 \leftarrow R_6 + R_5$ $[Y_3]$

$6M + 6S$:

$$\alpha = 3(X_1 + Z_1^2)(X_1 - Z_1^2), \qquad -\beta = -2X_1(2Y_1^2 + 2X_1) + (2X_1)^2,$$
$$\alpha\beta = \alpha(\alpha + \beta) - \alpha^2, \qquad Z_3 = 2Y_1 Z_1,$$
$$X_3 = (\alpha - \beta)(\alpha + \beta) + \beta^2 - 2\beta, \qquad Y_3 = \alpha\beta + (-\alpha)X_3 - 2(2Y_1^2)^2.$$

The resulting atomic blocks can be found in Table 9 with inputs $R_1 \leftarrow X_1$, $R_2 \leftarrow Y_1$, and $R_3 \leftarrow Z_1$, and output $X_3 \leftarrow R_1$, $Y_3 \leftarrow R_2$, $Z_3 \leftarrow R_3$.

Message-Based Traitor Tracing with Optimal Ciphertext Rate

Duong Hieu Phan[1,2], David Pointcheval[2], and Mario Strefler[2]

[1] LAGA, University of Paris 8
[2] ENS / CNRS / INRIA

Abstract. Traitor tracing is an important tool to discourage defrauders from illegally broadcasting multimedia content. However, the main techniques consist in tracing the traitors from the pirate decoders they built from the secret keys of dishonest registered users: with either a black-box or a white-box tracing procedure on the pirate decoder, one hopes to trace back one of the traitors who registered in the system. But new techniques for pirates consist either in sending the ephemeral decryption keys to the decoders for real-time decryption, or in making the full content available on the web for later viewing. This way, the pirate does not send any personal information. In order to be able to trace the traitors, one should embed some information, or watermarks, in the multimedia content itself to make it specific to the registered users.

This paper addresses this problem of tracing traitors from the decoded multimedia content or rebroadcasted keys, without increasing too much the bandwidth requirements. More precisely, we construct a message-traceable encryption scheme that has an optimal ciphertext rate, i. e. the ratio of global ciphertext length over message length is arbitrarily close to one.

1 Introduction

Traitor tracing (TT) [CFN94] is a cryptographic primitive used to broadcast content only to a set of authorized users, with an additional tracing property. Unlike broadcast encryption, the set of authorized users is fixed. The two main goals of such a primitive are

- confidentiality: only registered users have access to the broadcast content;
- traceability: if registered users share their secrets to allow unregistered users to access the content, one can trace back at least some of these traitors.

The former property is guaranteed by an encryption procedure, so that only registered users can decrypt and access the content. But an encryption scheme does not prevent users from giving away their secret keys. Even in case several users combine their secret keys in order to make a decryption box (a "pirate decoder"), it should be possible to identify one of the traitors from the code/secrets in the decoder (white-box tracing) or by simply interacting with the decoder (black-box tracing). The *tracing* property should indeed guarantee that even if several

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 56–77, 2012.

users collude to construct a pirate decoder, at least one of the traitors could be found. It should also guarantee non-frameability: an honest user should not be wrongly declared as a traitor.

To circumvent tracing, pirates might try not to make the decoder available, which excludes both white-box and black-box tracing. Instead, they could make only the decrypted content available, or, in case a hybrid encryption scheme is used, the symmetric keys used to encrypt the content: by *message-based traitor tracing*, we aim at tracing traitors from this information only, the decoded content.

Message-Based Traitor Tracing. We propose the term "message-based traitor tracing" as a generic term that subsumes earlier variants and emphasizes the fact that we do not trace from pirate decoders but from information embedded in the content. Fiat and Tassa were the first to consider message-based traitor tracing; in [FT99], they developed *dynamic traitor tracing* to deal with pirates that rebroadcast decrypted content. They assume that there is a real-time feedback from the broadcast content to the center, so that the watermarks can be adapted to the feedback. Safavi-Naini and Wang [SNW00] noted that in this setting, dynamic TT can be prevented by delaying the rebroadcasting of the content. To take this counter-measure into account, they proposed *sequential traitor tracing*, where the mark allocation is precomputed, but users are removed according to the feedback received. They construct a sequential TT scheme by combining error-correcting codes and watermarking. Jin and Lotspiech [JL07] claimed that protection should not increase the bandwidth by more than 10 %. To solve this problem, they proposed to extend the tracing procedure over several movies (using "inner" and "outer" codes) and assumed that the pirates will not drop any block. Their sequence key block scheme permits the revocation of users after they have been traced through the rebroadcasted messages. Kiayias and Pehlivanoglu [KP09] showed that the sequence key block scheme allows only to trace and to revoke a limited number of users, and proposed a message-trace-and-revoke scheme without this limitation.

Optimal Ciphertext Rate. Contrary to the classical tracing where schemes with optimal ciphertext rate exist, the problem of constructing a scheme with optimal ciphertext size for message-based traitor tracing is still open. We explain why the solutions for classical tracing fail when applied to message-based traitor tracing and we then describe our approach.

Boneh and Franklin [BF99] developed a traitor tracing scheme with a ciphertext size linear in the maximal number of colluding users. Kiayias and Yung [KY02] further integrated a 2-user version of this scheme with a fingerprinting code into the first TT scheme with a constant ciphertext rate. This method can be summarized as follows. The sender essentially encrypts all the blocks twice, so that the recipient can only decrypt one of the two ciphertexts for each block. The tracing procedure consists in using the decrypted ciphertext or the distributed keys to extract a word associated to the pirate decoder. Thanks to the tracing capability of a fingerprinting code, one can then trace back one of the traitors.

Kiayias and Yung's scheme leads to a ciphertext three times bigger than the initial content. Fazio, Nicolosi, and Phan [FNP07] then achieved a ciphertext rate asymptotically 1. Their method is to encrypt just one particular block twice each time and then apply an all-or-nothing transform (AONT), which guarantees that the pirate cannot drop this particular block because missing just one block makes the pirate unable to get any information on the plaintext. The use of AONT in [KY02, FNP07] is interesting but quite impractical because the receiver should wait until he has received n blocks (where n is the code length of the code in use, and thus quite large) to start the decryption procedure. We note that, without aiming to optimize the ciphertext rate, the use of AONT can be avoided by using robust fingerprinting code which allows pirate to drop a fraction of the positions. This is used in [Sir07, BP08, BN08] to reduce the ciphertext size. However, in order to get optimal ciphertext rate in [FNP07], the use of AONT is compulsory, otherwise the pirate could simply drop the particular block to defeat the tracing procedure.

Focusing now on message-based traitor tracing, one natural question is why we do not simply apply the above method of optimizing the ciphertext rate. We argue that this method cannot work for message-based traitor tracing. We first notice that in all the above methods for classical tracing, each user finally gets the same plaintext and if a user redistributes this plaintext, we have no way to trace back the traitor from the distributed message. Therefore, the necessary condition for message-based traitor tracing is that each user receives a different (marked) version of the plaintext. However, when the plaintext is different for each user, one cannot apply AONT for a whole fixed plaintext, otherwise all but at most one user can decrypt. The use of AONT for message-based traitor tracing is thus irrelevant. Fortunately, we can still use the method of doubling one particular block by finding out a way to hide this block. Our method consists in using a 2-user anonymous broadcast encryption scheme and then randomly permuting the blocks. With a 2-user anonymous broadcast encryption scheme, the pirate cannot detect any difference between an encryption for both users (which is used for all blocks but the particular block) and an encryption for one of the two users that is used for the particular protected block. Combining with the permutation of the blocks, we can show that the pirate is prevented from detecting the particular protected block. Moreover, beyond the optimization of ciphertext rate, by not using AONT, our scheme also enjoys the property of the sequential decryption via the use of fingerprinting code as in [BP08, BN08]: the user can sequentially decrypt the sub-ciphertexts, and does not need to wait to have received the whole ciphertext and to apply the AONT transform to start the decryption procedure.

Our Contribution. Our goal is to improve the technique which consists in distributing two versions of each message block, but without doubling all the blocks. The simplest way, presented in Section 3.1, is to have for each message block m_i two equivalent blocks m_i^0 and m_i^1, so that any sequence $\{m_i^{w_i}\}$, whatever $w \in \{0,1\}^n$ is, corresponds to a valid content m. The two versions m_i^0 and m_i^1 can be provided by either adding watermarks to the original message block

m_i, or directly, e. g. by recording a movie with different angles or distances of the shots [BS98]. The blocks, m_i^0 or m_i^1, are both sent over the public channel. However, the user secret keys, usk_i^0 or usk_i^1, have been distributed to the users according to codewords in a fingerprinting code. This means when the authority sees the decoded message m' or the symmetric keys, from each block m_i', it can tell whether it is m_i^0, m_i^1, or the block has been dropped, and then learns which decryption key has been used: usk_i^0, usk_i^1, or none. From this, it can derive one bit of a word: 0, 1, or 'erasure' respectively. Thanks to the collusion-resistance of the code with erasures, if not too many traitors colluded, at least one of them can be traced back. This method thus consists in encrypting each pair of blocks with two keys. Each user owns only one of the two according to the codeword he received from a fingerprinting code. This results in a ciphertext twice the length of the original message, plus the cost for two key encapsulations per block.

To reduce the length of the encrypted payload, the only way is to protect only a few blocks, not all of them. We present our basic construction in Section 3.2. It resists collusions, but only if the adversary has to output complete messages, i. e. he is not allowed to drop blocks.

If an adversary can detect which blocks are protected because he has access to several different user keys, he can drop some of them without impacting too much the quality of the original message (i. e. a few seconds from a movie). If the adversary can specifically drop protected blocks after decryption, the output contains less information about the keys that were used, which prevents tracing. We thus propose in Section 3.3 an extension of fingerprinting codes to solve this problem so that the adversary cannot specifically erase bits of the codeword: even if we protect 1% of the blocks and the adversary drops 20% of the blocks, he will basically drop only 20% of the protected blocks, and not all of them.

A further improvement, presented in Section 3.4 takes advantage of the fact that some public-key encryption schemes can reuse the randomness in both key encapsulations. For example in ElGamal, given g^r and two public keys $X_0 = g^{x_0}$ and $X_1 = g^{x_1}$, one can derive two sessions keys $Y_0 = X_0^r = g^{rx_0}$ and $Y_1 = X_1^r = g^{rx_1}$. This further reduces the number of group elements needed to encrypt.

This scheme still suffers from long user keys, as we need two key pairs for each message block. In Section 4, we use anonymous broadcast encryption as a primitive instead of PKE to achieve shorter key lengths. We first focus on the two-user case (one message block), which we later extend to cover N users. A message block either consists of a unique message m_i (not protected) or of two versions m_i^0 and m_i^1: in the former case, m_i should be encrypted for the two users, whereas in the latter case, m_i^0 has to be encrypted for user 0, and m_i^1 for user 1. To this aim, we use a 2-user anonymous broadcast encryption scheme (2ABE). Anonymous broadcast encryption allows the selection of any subset of the user set that should be able to decrypt the ciphertext, while hiding who is able to decrypt [LPQ11]. Suppose we have a 2ABE scheme, and we consider ℓ blocks $(m_1, \ldots, m_\ell)$, among which the k-th block only is protected and thus is provided as a pair (m_k^0, m_k^1). We encrypt all the unique blocks m_i for both users, whereas we encrypt m_k^0 for user 0, and m_k^1 for user 1. The ciphertexts are thereafter

randomly permuted (but we assume that the message blocks contain indices to reorder them). User 0 and user 1 will both be able to decrypt ℓ ciphertexts among the $\ell + 1$, and after reordering will be able to get the original message. Due to the anonymity, they do not know which block the other user cannot decrypt, therefore they have no idea which block is protected.

The encrypted payload is only $(1 + 1/\ell)$-times as long as the original message, plus the cost of 2ABE key encapsulations, but we need only one key for each user, which is the minimum, given that when viewing the decrypted message, the authority can extract one bit. To achieve full tracing, we extend this case to an arbitrary number of users by allocating the user secret keys for the 2ABE using our extension of a fingerprinting code [BS98].

2 Definitions

In this section, we define message-based traitor tracing schemes and the building blocks we will use in their construction. We follow an approach similar to [NSS99] by defining first a two-user primitive which we then extend to the multi-user case using fingerprinting codes.

We first state the *marking assumption*, which provides a way to embed a bit in a message block. This will be applied to blocks we protect, whereas no bit will be embedded in non-protected blocks. Then, from the decoded message, the authority will be able to extract the bits involved in the decryption keys in the pirate decoder, unless the decoder drops the protected blocks. We will thus need the property that nobody can detect which blocks are protected so that if the pirate decoder decides to drop some blocks, the choice will be independent from the protection of the blocks. We will show that we can build such a *message-traceable encryption* from a *2-user anonymous broadcast encryption* scheme. Eventually, from all the bits extracted from the protected blocks (and erasures in case of dropped blocks), using the tracing algorithm of a *fingerprinting code*, we can trace back some of the traitors.

2.1 Primitives

As usual, a public-key encryption scheme is defined as a 4-tuple of algorithms $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, along with the standard security notion IND-CPA. A more formal definition of PKE is given in Section A.1 of the appendix. We define an anonymous broadcast encryption scheme as a 3-tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Encaps}, \mathsf{Decaps})$, along with the standard security notion IND-ACPA. A more formal definition of ABE is given in Section A.2 of the appendix.

2.2 Marking Content

In order to trace from the message content itself, we need to be able to distribute different versions of a message to different users in an undetectable way. One

way is to use watermarks. Another way could exploit different camera shots (angle and distance) of the same scene in a movie [BS98]. We abstract away from the concrete way to create versions and use the *marking assumption* that has been introduced by [BS98] to abstract from concrete schemes and has become standard since. [KP10] In the following we assume that, given two blocks m_0 and m_1,

- we can double m_b (for a random $b \in \{0,1\}$) into two *equivalent* messages m_b^0 and m_b^1
- when a user receives m'_0 and m'_1 (such that $m'_b \in \{m_b^0, m_b^1\}$ and $m'_{\bar{b}} = m_{\bar{b}}$, where $\bar{b} = 1 - b$), he cannot guess b.

This essentially means that it is possible to mark a message to protect it, but it is not possible to tell apart protected and unprotected blocks.

In addition, we also assume *robustness* with respect to a symmetric, reflexive relation $\approx_\rho$: for two equivalent blocks $m^0 \approx_\rho m^1$, when the user receives m^b, and tries to alter it (but without changing the meaning or content), he has only a negligible chance to output $m' \approx_\rho m^b$ that is closer to $m^{\bar{b}}$ than to m^b. This reflects that the user cannot change a watermark while preserving the message.

The robustness and the marking assumption guarantee that

- a protected block is indistinguishable from an unprotected block;
- when a user has access to one version of the protected block only, we can learn from its output which bit was embedded: the *detected bit*;
- when a user has access to both versions of the protected block, we either detect from its output one explicit bit as above, or we note that both versions have been used: in either case we can output one bit, associated to at least one version of the block available to the user.

Of course, the user can drop some blocks, but this impacts the quality of the message: we will assume that at most a fraction η of the blocks are dropped.

2.3 Fingerprinting Codes

Fingerprinting codes [BS98] allow an authority to trace a subset of the users (the traitors) that colluded to produce a word (pirate word) from the codewords they were given. This of course depends on the way traitors can derive words from their codewords: the *feasible set* is the set of the useful words that can be derived from the legitimate codewords. We focus on binary codes, defined over the alphabet $\{0,1\}$. In our context, each bit-value is associated to a decryption key, and a receiver has to decrypt at least one block in each pair of variants to be able to get the global content. If all the codewords in a set agree on a position (i. e. they all have the same bit at this position), then the collusion owns only one decryption key, and thus all the words in the feasible set must have the same bit at this position. However, if some of the words differ at a given position, then the collusion owns both decryption keys, and thus both values are possible at this position. More formally, for any list of t words $w_1, \ldots, w_t \in \{0,1\}^n$, $\mathcal{FS}(w_1, \ldots, w_t) = \{w \in \{0,1\}^n \,|\, \forall i \in \{1, \ldots, n\}, \exists j \in \{1, \ldots, t\}, w[i] = w_j[i]\}$.

Definition 1. *A* t-fingerprinting code $\mathcal{T}$ *for* $\mathcal{FS}$ *is defined by a pair of algorithms* $(\mathsf{Gen}, \mathsf{Trace})$*, where*

- $\mathsf{Gen}(N, \varepsilon)$ *takes as input the number* N *of codewords to output and an error probability* ε*, it outputs a tracing key* tk *and a code* $\Gamma \subset \{0,1\}^n$ *of size* N*.*
- $\mathsf{Trace}(\mathsf{tk}, w')$ *takes as input the tracing key* tk *and a word* $w' \in \mathcal{FS}(C)$*, where* C *is a collusion of at most* t *codewords, it outputs a codeword* w*.*

The running time of both algorithms must be polynomial in $N \log(1/\varepsilon)$*, and the tracing algorithm should not be wrong too often: with probability less than* ε*,* $w \notin C$*.*

More precisely, a t-fingerprinting code for $\mathcal{FS}$ guarantees that

- given $(\Gamma, \mathsf{tk}) \leftarrow \mathsf{Gen}(N, \varepsilon)$, with $\Gamma \subset \{0,1\}^n$ of size N
- for any collusion $C \subset \Gamma$ of size at most t, for any $w \in \mathcal{FS}(C)$, $\mathsf{Trace}(\mathsf{tk}, w)$ outputs a word in C with probability $1 - \varepsilon$.

Efficient constructions of such codes can be found in [Tar08]: the resulting code length n for a t-collusion-resistant fingerprinting code is $\mathcal{O}(t^2 \log(N/\varepsilon))$, where ε is the tracing error probability.

2.4 Message-Traceable Encryption

A message-traceable encryption scheme Ψ is a multi-cast encryption scheme which allows all the registered users (with a legitimate secret key) to decrypt a ciphertext. In addition, from the decrypted content, it is possible to derive the key (or even the keys) used for the decryption. In the following description, we focus on static schemes (the maximum number of users is set from the beginning):

- $\mathsf{Setup}(1^\kappa, N, t, \varepsilon)$, where κ is the security parameter, N the number of users, t the maximal size of a collusion, and ε the error probability of the tracing algorithm, generates the global parameters param of the system (omitted in the following), N user secret keys $\{\mathsf{USK}_{\mathsf{id}}\}_{\mathsf{id}=1,\ldots,N}$, an encryption key EK, and a tracing key TK.
- $\mathsf{Encrypt}(\mathsf{EK}, m)$ takes as input the encryption key EK and a message m to be sent, it generates a ciphertext c.
- $\mathsf{Decrypt}(\mathsf{USK}, c)$ takes as input a decryption key USK and a ciphertext c, it outputs a message m, or the error symbol $\perp$.
- $\mathsf{Trace}(\mathsf{TK}, c, m)$ takes as input the tracing key TK, a ciphertext c and the decrypted message m, returns an index $\mathsf{id} \in [1, N]$ of a user secret key $\mathsf{USK}_{\mathsf{id}}$.

Security Notions. As for any encryption scheme, the first security notion to define is semantic security, in our case against chosen-ciphertext attacks, whose security game is presented in Figure 1. Of course, to make tracing possible, the encryption algorithm will possibly derive several equivalent versions of the message m_b to be encrypted, which will decrypt to slightly different messages depending on the key used to decrypt. For this reason, we allow the adversary to choose which decryption key should be used by the decryption oracle, hence the additional input id.

$\mathrm{Exp}^{\mathsf{ind-cca}-b}_{\Psi,\mathcal{A}}(\kappa, N, t, \varepsilon)$
$(\{\mathsf{USK}_{\mathsf{id}}\}, \mathsf{EK}, \mathsf{TK}) \leftarrow \mathsf{Setup}(1^\kappa, N, t, \varepsilon)$; $\mathcal{Q}_D \leftarrow \emptyset$;
$(state, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{ODecrypt}(\cdot,\cdot)}(\mathsf{FIND}; \mathsf{EK})$;
$c^* \leftarrow \mathsf{Encrypt}(\mathsf{EK}, m_b)$;
$b' \leftarrow \mathcal{A}^{\mathsf{ODecrypt}(\cdot,\cdot)}(\mathsf{GUESS}; state, c^*)$;
`if` $c^* \in \mathcal{Q}_D$ `then return` 0 `else return` b';

$\mathsf{ODecrypt}(\mathsf{id}, c)$
$\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{c\}$;
$m \leftarrow \mathsf{Decrypt}(\mathsf{USK}_{\mathsf{id}}, c)$;
`return` m;

Fig. 1. IND-CCA for message-traceable encryption

Definition 2 (Semantic Security). *A message-traceable encryption scheme Ψ is said to be $(\tau, N, t, \varepsilon, q_D, \nu)$-IND-CCA-secure (indistinguishability against chosen-ciphertext attacks) if in the security game presented in Figure 1, the advantage, denoted $\mathrm{Adv}^{\mathsf{ind-cca}}_{\Psi}(\kappa, \tau, N, t, \varepsilon, q_D)$, of any τ-time adversary $\mathcal{A}$ asking for at most q_D decryption queries (*ODecrypt *oracle) is bounded by ν.*

$$\mathrm{Adv}^{\mathsf{ind-cca}}_{\Psi}(\kappa, \tau, N, t, \varepsilon, q_D) = \max_{\mathcal{A}}\{\Pr[\mathrm{Exp}^{\mathsf{ind-cca}-1}_{\Psi,\mathcal{A}}(\kappa, N, t, \varepsilon) = 1] - \Pr[\mathrm{Exp}^{\mathsf{ind-cca}-0}_{\Psi,\mathcal{A}}(\kappa, N, t, \varepsilon) = 1]\}.$$

This definition includes IND-CPA (for Chosen-Plaintext Attacks) when $q_D = 0$, and thus we denote the advantage $\mathrm{Adv}^{\mathsf{ind-cpa}}_{\Psi}(\kappa, \tau, N, t)$.

We now formalize the additional security notion of *traceability*: after having received at most t secret keys (the collusion C of traitors), the adversary asks for a ciphertext $c^\star$ of a random message $m^\star$, and outputs a plaintext m that should be equivalent to $m^\star$. The tracing algorithm should then output one of the traitors, otherwise the adversary has won the game. We use the relation $m \approx_\rho m'$ from Section 2.2 to denote that two messages are "similar" in the current context. If the adversary sends a random message (hence $m \not\approx_\rho m^\star$) or alternatively outputs an empty message, we say the adversary lost the game:

Definition 3 (Traceability). *A message-traceable encryption scheme Ψ is said to be $(\tau, N, t, \varepsilon, \nu)$-traceable if in the security game presented in Figure 2, the success probability, denoted $\mathsf{Succ}^{\mathsf{trace}}_{\Psi,\mathcal{A}}(\kappa, \tau, N, t, \varepsilon)$, of any τ-time adversary asking for at most t secret keys is bounded by ν.*

$$\mathsf{Succ}^{\mathsf{trace}}_{\Psi}(\kappa, \tau, N, t, \varepsilon) = \max_{\mathcal{A}}\{\Pr[\mathrm{Exp}^{\mathsf{trace}}_{\Psi,\mathcal{A}}(\kappa, N, t, \varepsilon) = 1]\}.$$

3 A Generic Construction from PKE

In this section, we present a series of three simple constructions that illuminate the concepts behind the construction in the next section. The first construction exemplifies a well-known paradigm of constructing a message-based traitor tracing scheme from a PKE scheme and a fingerprinting code, and serves as a stepping stone for the second construction, which shows how to achieve optimal

$\mathrm{Exp}^{\mathsf{trace}}_{\Psi,\mathcal{A}}(\kappa, N, t, \varepsilon)$
$(\{\mathsf{USK}_{\mathsf{id}}\}, \mathsf{EK}, \mathsf{TK}) \leftarrow \mathsf{Setup}(1^\kappa, N, t, \varepsilon)$; $\mathcal{Q}_C \leftarrow \emptyset$;
$(state, C) \leftarrow \mathcal{A}(\mathsf{FIND}; \mathsf{EK})$;
$m^\star \xleftarrow{\$} \mathcal{M}$; $c^\star \leftarrow \mathsf{Encrypt}(\mathsf{EK}, m^\star)$;
$m \leftarrow \mathcal{A}(\mathsf{GUESS}; state, c^\star, \{\mathsf{USK}_{\mathsf{id}}\}_{\mathsf{id}\in C})$;
$T \leftarrow \mathsf{Trace}(\mathsf{TK}, c^\star, m)$;
`if` $m = \perp$ `or` $m \not\approx_\rho m^\star$ `then return` 0;
`if` $T \cap C = \emptyset$ `then return` 1 `else return` 0;

Fig. 2. Traceability

ciphertext rate. The first two constructions are only secure when the adversary is required to retransmit only complete messages. Our third construction modifies the previous one to account for adversaries that drop parts of the message. The fourth construction is an exercise to drive the efficiency of the second construction to its limits by reusing randomness.

3.1 A Simple Construction

To have a baseline against which to compare our later constructions, we first outline a simple way to construct a message-based traitor tracing scheme. The construction uses a PKE scheme Π and assigns user keys according to the codewords of the fingerprinting code $\mathcal{T}$. If the codewords have length n, we need $2n$ instances of the PKE scheme.

- $\mathsf{Setup}(1^\kappa, N, t, \varepsilon)$
  1. It generates a t-fingerprinting code, using $(\Gamma, \mathsf{tk}) \leftarrow \mathcal{T}.\mathsf{Gen}(N, \varepsilon)$, with a low error value ε. We thus denote n the length of a codeword in Γ, and enumerate codewords with indices associated to each users: $\Gamma = \{w_{\mathsf{id}}\}_{\mathsf{id}=1,\ldots,N} \subset \{0,1\}^n$.
  2. it then calls $2n$ times $\Pi.\mathsf{Setup}(1^\kappa)$ to obtain $(\mathsf{dk}_i^b, \mathsf{ek}_i^b)_{b=0,1,i=1\ldots n}$.
  3. it sets $\mathsf{EK} \leftarrow \{\mathsf{ek}_i^b\}_{b=0,1,i=1\ldots n}$, $\mathsf{USK}_{\mathsf{id}} \leftarrow (\mathsf{dk}_i^{w_{\mathsf{id}}[i]})_{i=1\ldots n}$ for all $\mathsf{id} \in [1, N]$, $\mathsf{TK} \leftarrow (\{\mathsf{dk}_i^0, \mathsf{dk}_i^1\}_{i=1,\ldots,n}, \mathsf{tk})$.
- $\mathsf{Encrypt}(\mathsf{EK}, m)$ first splits m into n blocks $m_1, \ldots m_n$. For each block,
  1. it creates two versions m_i^0, m_i^1 of each block m_i
  2. it then encrypts the versions as $c_i^b = \Pi.\mathsf{Encrypt}(\mathsf{ek}_i^b, m_i^b)$
  3. it sets $c = (c_1^0, c_1^1, \ldots, c_n^0, c_n^1)$.
- $\mathsf{Decrypt}(\mathsf{USK}_{\mathsf{id}}, c)$
  1. it parses $c = (c_1^0, c_1^1, \ldots, c_n^0, c_n^1)$
  2. it decrypts $m_i^{w_{\mathsf{id}}[i]} = \Pi.\mathsf{Decrypt}(\mathsf{dk}_i^{w_{\mathsf{id}}[i]}, c_i^{w_{\mathsf{id}}[i]})$ to recover m.
- $\mathsf{Trace}(\mathsf{TK}, c, m)$ extracts the word w' from m and calls $\mathcal{T}.\mathsf{Trace}(\mathsf{tk}, w')$ to get the codeword w_{id} of a colluder.

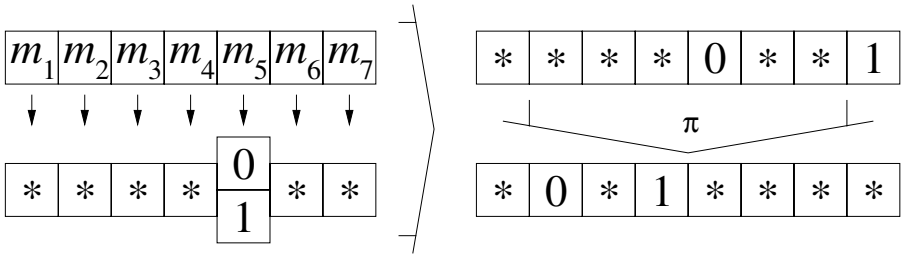

Fig. 3. Hiding a mark at position 5 in a sequence of 7 blocks

3.2 Improved Construction

We can reduce the ciphertext rate for long messages by watermarking only some blocks. We now describe a generic construction that accomplishes this by encrypting a message consisting of n sequences of ℓ blocks each in such a way that in sequence i, $\ell - 1$ blocks can be decrypted by both users; these blocks are not used for tracing. The other block is duplicated using to different marks and encrypted at two positions $v_0[i], v_1[i]$, each time for one key only: the message at position $v_0[i]$ cannot be decrypted by users with key 0, and the message at position $v_1[i]$ cannot be decrypted by users with key 1. By doing this, the ciphertext will have a length of $(1+1/\ell)$-times the length of the message, plus the overhead for encryption.

To reduce the overhead for encryption for long messages, we now model the PKE scheme as a KEM. Given any symmetric cipher $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$, a PKE Π, and a fingerprinting code $\mathcal{T}$, we construct a message-traceable encryption scheme $\hat{\Psi}(\ell, n)$ as follows:

- $\mathsf{Setup}(1^\kappa, N, t, \varepsilon, \ell)$:
  1. It generates a t-fingerprinting code, using $(\Gamma, \mathsf{tk}) \leftarrow \mathcal{T}.\mathsf{Gen}(N, \varepsilon)$, with a low error value ε. We thus denote n the length of a codeword in Γ, and enumerate codewords with indices associated to each users: $\Gamma = \{w_{\mathsf{id}}\}_{\mathsf{id}=1,\ldots,N} \subset \{0,1\}^n$.
  2. It then calls $\Pi.\mathsf{Setup}(1^\kappa)$ $n(\ell+1)$ times to obtain, for $i = 1, \ldots, n$ and $j = 1, \ldots, \ell+1$, $\mathsf{ek}_{i,j}, \mathsf{dk}_{i,j}$. It draws two random vectors $v_0, v_1 \in [1, \ell+1]^n$ with the condition that $v_0[i] \neq v_1[i]$ for all $i = 1, \ldots n$. The position $v_b[i]$ describes the secret key that the users with $w_{\mathsf{id}}[i] = b$ do *not* have. We set

$$\begin{aligned}
\mathsf{USK}_{\mathsf{id}} &\leftarrow \{\mathsf{dk}_{i,j}\}_{\substack{i=1,\ldots,n \\ j \neq v_{w_{\mathsf{id}}[i]}[i]}}, \\
\mathsf{EK} &\leftarrow (\{\mathsf{ek}_{i,j}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,\ell+1}}, v_0, v_1), \\
\mathsf{TK} &\leftarrow (\{\mathsf{dk}^0_{i,j}, \mathsf{dk}^1_{i,j}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,\ell+1}}, \mathsf{tk}).
\end{aligned}$$

- $\mathsf{Encrypt}(\mathsf{EK}, m)$ first splits m in $n\ell$ blocks $\{m_{i,j}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,\ell}}$. For each sequence, at position $i \in \{1, \ldots, n\}$:

1. it chooses a random position $k \in \{1, \ldots, \ell\}$, to protect block $m_{i,k}$;
2. it generates two equivalent versions $m^0_{i,k}, m^1_{i,k}$, of this block (see Figure 3), resulting in a list of $\ell + 1$ blocks;
3. it prepends the position to the block: $M_j = j \| m_{i,j}$, for $j = 1, \ldots, \ell$, $j \neq k$, $M_k = k \| m^0_{i,k}$, $M_{\ell+1} = k \| m^1_{i,k}$;
4. it chooses a random permutation $\pi \in S_{\ell+1}$ with the restriction that the position of the marked blocks is $v_1[i] = \pi(k)$ and $v_0[i] = \pi(\ell + 1)$; and permutes the blocks: $M'_j = M_{\pi(j)}$.
5. it generates the session keys: $(c_{i,j}, K_{i,j}) \leftarrow \Pi.\mathsf{Encaps}(\mathsf{ek}_{i,j})$,
6. It then encrypts the blocks under the symmetric keys: $C_{i,j} \leftarrow (c_{i,j}, c'_{i,j} = \mathsf{Enc}_{K_{i,j}}(M'_j))$ for $j = 1, \ldots, \ell + 1$.

The final ciphertext consists of all the pairs $C_{i,j}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, \ell + 1$.

– $\mathsf{Decrypt}(\mathsf{USK}_{\mathsf{id}}, C)$ takes as input the key $\mathsf{USK}_{\mathsf{id}} = \{\mathsf{dk}_{i,j}\}_{\substack{i=1,\ldots,n \\ j \neq v_{w_{\mathsf{id}}[i]}[i]}}$ and a ciphertext $C = \{C_{i,j}\}$. For each sequence, at position $i \in \{1, \ldots, n\}$:
   1. it calls $\Pi.\mathsf{Decaps}(\mathsf{usk}^{w_{\mathsf{id}}[i]}_{i,j}, c_{i,j})$, for $j \neq v_{w_{\mathsf{id}}[i]}[i]$, to obtain the session key $K_{i,j}$;
   2. it decrypts the message with $\mathsf{Dec}(K_{i,j}, c'_{i,j})$, which outputs $M'_{i,j}$;
   3. it should be able to parse the $M'_{i,j} = p_j \| m_{i,j}$, with $\{p_j\} = \{1, \ldots, \ell\}$, otherwise it stops and outputs $\perp$;
   4. it eventually reorders the messages according to p_j, and concatenates the other parts.

   It concatenates the ℓ blocks in each sequences, and the n sequence-results to output the full message m.

– $\mathsf{Trace}(\mathsf{TK}, C, m)$ can detect the protected blocks using the decryption keys in TK. From the block that was actually decrypted in each sequence i, it can learn the value of the bit $w[i]$. Then, thanks to the traceability of the code $\mathcal{T}$, the $\mathcal{T}.\mathsf{Trace}(\mathsf{tk}, w)$ outputs a traitor.

Remark 4. The traceability of the scheme rests on the fact that a user does not know which of the keys are common to all users and which are specific to those with the same bit in the codeword. While a user that shares the information which positions he cannot decrypt with other users is considered to be misbehaving and thus corrupted in our security model, the real-life cost of sharing some of these positions is quite low. The scheme is thus susceptible to a Pirates 2.0-attack as described in [BP09].

3.3 Adapting the Code for Deletions

The above scheme works well if we require that the users output only complete messages. However, in practice removing small parts of a movie might still result in an acceptable quality. In this case, we want to ensure that the adversary cannot drop specifically the blocks that contain watermarks, but that in order to delete a certain fraction of the codeword, he has to delete the same fraction of blocks. If users collude in constructing the message, i. e. the adversary has access to

several user secret keys, they can choose to drop only the blocks that contain watermarks. In this case, in the places where the codewords that correspond to the user secret keys differ, the adversary can find the marked message blocks and drop only those. The tracing authority cannot know if a block was dropped at random or because the adversary knew it was marked, so tracing is impossible.

To solve this problem, we design a new code from a fingerprinting code by repeating and permuting the bits of the codeword. If the adversary cannot tell which bits of the new code are repetitions of the same bit, it is unlikely that all repetitions of a bit from the fingerprinting code are erased accidentally. The tracing authority can then assume that for an erased bit, both keys are known to the attacker. The only problem is that colluding users can detect in which sequence they all decrypt the same blocks, which implies that they have the same code bit in this position. They can then drop blocks specifically from sequences where their code bits agree. To prevent this, we insert dummy bits that are the same for all users, which doubles the length of the code.

Let η be the fraction of blocks that the adversary is allowed to drop. We choose an integer ρ such that $\eta^\rho \leq \varepsilon/2n$. Let $w = w[1] \dots w[n]$ be a codeword from the fingerprinting code. We generate a codeword of our new code by repeating each bit of w ρ times, padding it with $n\rho$ dummy bits that are identical for all users, then applying a permutation:
We first describe $\mathsf{Gen}(N, \varepsilon)$:

1. Generate the fingerprinting code: $(\Gamma, \mathsf{tk}) \leftarrow \mathcal{T}.\mathsf{Gen}(N, \varepsilon/2)$
2. Choose a random permutation $\pi : \{0,1\}^{2n\rho} \rightarrow \{0,1\}^{2n\rho}$
3. $\mathsf{tk}' \stackrel{\mathsf{def}}{=} (\mathsf{tk}, \pi^{-1})$
4. Choose a random string $s \stackrel{\$}{\leftarrow} \{0,1\}^{n\rho}$
5. $\Gamma' \stackrel{\mathsf{def}}{=} \{w'_{\mathsf{id}}\}_{\mathsf{id}=1,\dots,N} \subset \{0,1\}^{2n\rho}$ where $w'_{\mathsf{id}} \stackrel{\mathsf{def}}{=} \pi(w_{\mathsf{id}}[1] \dots w_{\mathsf{id}}[1] \dots w_{\mathsf{id}}[n] \| s)$.
6. Output (Γ', tk').

To trace, run the modified algorithm $\mathsf{Trace}(\mathsf{tk}', w')$:

1. Reconstruct a word w from $\pi^{-1}(w')$:
   (a) If all of the ρ replications of bit $w[i]$ that were not erased are equal to b, set $w[i] = b$.
   (b) If at least one of the replications of bit $w[i]$ has the value 1 and at least one of them has the value 0, choose $w[i]$ at random (in this case the adversary knows both keys).
   (c) If all replications of the bit $w[i]$ have been erased, choose $w[i]$ at random.
2. Return the output of $\mathcal{T}.\mathsf{Trace}(\mathsf{tk}, w)$.

Since a codeword is 2ρ times as long as previously, the user secret keys will also be 2ρ times as long. The tracing key tk contains a secret permutation π^{-1}, so tracing is no longer public as opposed to tracing for the fingerprinting code. However, in our construction we need to include the user secret keys in the tracing key, so in our construction tracing was not public even before the change in the code used.

3.4 Reusing Randomness

To further reduce the ciphertext rate, we can try to reduce the size of the key encapsulation, by reusing the random coins in all the ciphertexts of a sequence or even the complete ciphertext.

We need to make sure that the PKE scheme in the construction remains secure if randomness us reused. If we instantiate the PKE scheme with ElGamal, we can use the results of Bellare, Boldyreva, Kurosawa, and Staddon [BBKS07, Lemma 7.2].

The only change is in the encryption:

– $\mathsf{Encrypt}(\mathsf{EK}, m)$: it first splits m in $n\ell$ blocks $\{m_{i,j}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,\ell}}$. For each sequence, at position $i \in \{1, \ldots, n\}$:

  5. it draws the common randomness $R \xleftarrow{\$} \mathcal{R}$, then generates the session keys: $(c_{i,j}, K_{i,j}) \leftarrow \Pi.\mathsf{Encaps}(\mathsf{ek}_{i,j}; R)$,

The final ciphertext consists, as above, of all the pairs $C_{i,j} = (c_{i,j}, c'_{i,j})$, but where all the $c_{i,j}$ in the same sequence use a common part that can be included once only.

Reusing randomness incurs a loss in the security reduction for the PKE scheme that is equal to the number of ciphertexts that share the same randomness [BBKS07, Th. 6.2]. This means that in the generic case, the group has to be larger to accommodate the loss in security. In the case of ElGamal however, we can exploit the random self-reducibility to avoid this increase in the group size [BBKS07, Th. 7.3].

3.5 Security

For reasons of space, and since these constructions are not the main result of this paper, we do not give a security proof for them. Given the security proof of our final construction, the security of the above constructions is an easy corollary.

4 A Construction with Shorter Keys

The main disadvantage of the PKE-based construction is the length of the user keys, which must contain a PKE key for each block. Since a codeword has n bits, we can hope to reduce the number of different user keys to n as well. To achieve this, we use a primitive that allows encryption to either of two users or to both of them: 2-user broadcast encryption.[1]

Our message-traceable encryption scheme makes use of codes, where the bits of the codewords are embedded in a message by doubling some parts of it, the so-called *protected blocks*. Because we do not want the adversary to learn which parts of the message contain bits of the codeword, we need a broadcast

[1] We view the scheme as an anonymous broadcast encryption scheme because broadcast encryption is a well-known concept that is intuitively understood. In [KY02], the same primitive was called a "2-key, 1-copyrighted public-key encryption scheme".

encryption scheme where a user cannot tell whether a block is destined only for his key or for both keys, a 2-user anonymous broadcast encryption (2ABE).

This requires the symmetric cipher used with this construction to be weakly robust [ABN10], since one of the decapsulated keys will be either $\perp$ or an unusable key. The construction uses one instance of the 2ABE scheme Π per bit of the codeword, encrypting $\ell + 1$ messages at a time in one sequence, with the target sets determined by the positions v, w where the watermarks are embedded. In this construction, the length of the EK and USK is n times that of Π, and to encrypt a sequence of ℓ blocks, doubling one block, we need $\ell + 1$ Π key-encapsulations plus $\ell + 1$ symmetrically encrypted message blocks.

4.1 Construction of a Message-Traceable Encryption Scheme

Our first construction combines a fingerprinting code $\mathcal{T}$ with a 2ABE scheme Π. If the codewords have length n, we need n instances of the 2ABE scheme. Given any weakly robust [ABN10] symmetric cipher $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$, a 2-user anonymous broadcast encryption Π, and a traceable code $\mathcal{T}$, we construct a message-traceable encryption scheme $\Psi(\ell, n)$ as follows:

- $\mathsf{Setup}(1^\kappa, N, t, \varepsilon, \ell)$:
  1. It first generates a t-fingerprinting code, using $(\Gamma, \mathsf{tk}) \leftarrow \mathcal{T}.\mathsf{Gen}(N, \varepsilon)$. We thus denote n the length of a codeword in Γ, and enumerate codewords with indices associated to each users: $\Gamma = \{w_{\mathsf{id}}\}_{\mathsf{id}=1,\ldots,N} \subset \{0,1\}^n$.
  2. It then calls $\Pi.\mathsf{Setup}(1^\kappa, 2)$ n times to obtain, for $i = 1, \ldots, n$, $\mathsf{ek}_i, \mathsf{usk}_i^0, \mathsf{usk}_i^1$.
     We set

$$\begin{aligned} \mathsf{USK}_{\mathsf{id}} &\leftarrow (\mathsf{usk}_1^{w_{\mathsf{id}}[1]}, \ldots, \mathsf{usk}_n^{w_{\mathsf{id}}[n]}), \\ \mathsf{EK} &\leftarrow (\mathsf{ek}_1, \ldots, \mathsf{ek}_n), \\ \mathsf{TK} &\leftarrow (\{\mathsf{usk}_i^0, \mathsf{usk}_i^1\}_{i=1,\ldots,n}, \mathsf{tk}). \end{aligned}$$

- $\mathsf{Encrypt}(\mathsf{EK}, m)$: it first splits m in $n\ell$ blocks $\{m_{i,j}\}_{\substack{i=1,\ldots,n \\ j=1,\ldots,\ell}}$. For each sequence, at position $i \in \{1, \ldots, n\}$:
  1. it chooses a random position $k \in \{1, \ldots, \ell\}$, to protect block $m_{i,k}$;
  2. it generates two equivalent versions $m_{i,k}^0, m_{i,k}^1$, of this block (see Figure 3), resulting in a list of $\ell + 1$ blocks;
  3. it prepends the position to the block: $M_j = j \| m_{i,j}$, for $j = 1, \ldots, \ell$, $j \neq k$, $M_k = k \| m_{i,k}^0$, $M_{\ell+1} = k \| m_{i,k}^1$;
  4. it chooses a random permutation $\pi \in S_{\ell+1}$ and permutes the blocks: $M'_i = M_{\pi(i)}$. We note $v = \pi(k)$ and $w = \pi(\ell + 1)$, the positions of the two equivalent blocks;
  5. it generates session keys for all the blocks, except M'_v and M'_w, with the 2ABE scheme Π, with the full target set $\{0, 1\}$, whereas M'_v is targeted to $\{0\}$ only, and M'_w is targeted to $\{1\}$ only. More precisely, it first generates the session keys: $(c_{i,j}, K_{i,j}) \leftarrow \Pi.\mathsf{Encaps}(\mathsf{ek}_i, \{0,1\})$, for $j \neq v, w$, $(c_{i,v}, K_{i,v}) \leftarrow \Pi.\mathsf{Encaps}(\mathsf{ek}_i, \{0\})$, and $(c_{i,w}, K_{i,w}) \leftarrow \Pi.\mathsf{Encaps}(\mathsf{ek}_i, \{1\})$.

6. It then encrypts the blocks under the symmetric keys: $C_{i,j} \leftarrow (c_{i,j}, c'_{i,j} = \mathsf{Enc}_{K_{i,j}}(M'_j))$ for $j = 1, \ldots, \ell+1$.

The final ciphertext consists of all the pairs $C_{i,j}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, \ell+1$.

– $\mathsf{Decrypt}(\mathsf{USK}_{\mathsf{id}}, C)$ takes as input the key $\mathsf{USK}_{\mathsf{id}} = (\mathsf{usk}_1^{w_{\mathsf{id}}[1]}, \ldots, \mathsf{usk}_n^{w_{\mathsf{id}}[n]})$ and a ciphertext $C = \{C_{i,j}\}$. For each sequence, at position $i \in \{1, \ldots, n\}$:
  1. it calls $\Pi.\mathsf{Decaps}(\mathsf{usk}_i^{w_{\mathsf{id}}[i]}, c_{i,j})$, for $j = 1, \ldots, \ell+1$, to obtain the session key $K_{i,j}$;
  2. it decrypts the message with $\mathsf{Dec}(K_{i,j}, c'_{i,j})$, which outputs either $M'_{i,j}$ or $\perp$ (because of the robustness);
  3. it should be able to parse the $M'_{i,j} = p_j \| m_{i,j}$, with $\{p_j\} = \{1, \ldots, \ell\}$, otherwise it stops and output $\perp$;
  4. it eventually reorders the messages according to p_j, and concatenates the other parts.

  It concatenates the ℓ blocks in each sequences, and the n sequence-results to output the full message m.

– $\mathsf{Trace}(\mathsf{TK}, C, m)$ can detect the protected blocks using the decryption keys in TK. From the block that was actually decrypted in each sequence i, it can learn the value of the bit $w[i]$. Then, thanks to the traceability of the code $\mathcal{T}$, the $\mathcal{T}.\mathsf{Trace}(\mathsf{tk}, w)$ outputs a traitor.

4.2 Security of the Construction

We show that our construction fulfills IND-CPA-security and explain under which conditions it is traceable.

Theorem 5. *If the 2ABE scheme Π is IND-CPA and the symmetric encryption scheme $\mathcal{E}$ is IND-CPA, then our construction $\Psi(\ell, n)$ is IND-CPA, and*

$$\mathrm{Adv}^{\mathsf{ind-cpa}}_{\Psi(\ell,n)}(\kappa, \tau, N, t, \varepsilon) \leq n \cdot (\ell+1) \times \left(2 \cdot \mathrm{Adv}^{\mathsf{ind-cpa}}_{\Pi}(\kappa, \tau_1, 2) + \mathrm{Adv}^{\mathsf{ind-cpa}}_{\mathcal{E}}(\kappa, \tau_2)\right).$$

The proof can be found in the full version [PPS12].

Before we turn to tracing, we state an intermediate result. The following lemma says that no adversary can tell which blocks are not encrypted to all users, if he only has one of the two keys, w.l.o.g. usk_i^0.

Lemma 6. *If the 2ABE scheme Π is both IND-CPA and ANO-CPA, and the symmetric encryption scheme $\mathcal{E}$ is IND-CPA, then an adversary who only has the usk_i^0 for a sequence i cannot distinguish between the case where the block at position v is encrypted to the target set $\{0\}$ and the block at position w is encrypted to the target set $\{1\}$ and the case where the block at position v is encrypted to the target set $\{0, 1\}$ and the block at position w contains a random message. If we denote by $\mathrm{Adv}^{\mathsf{find}}_{\Psi}(\kappa, \tau, N, t)$ the maximal advantage of any adversary within time τ, on any index i, then*

$$\mathrm{Adv}^{\mathsf{find}}_{\Psi}(\kappa, \tau, N, t, \varepsilon) \leq \mathrm{Adv}^{\mathsf{ano-cpa}}_{\Pi}(\kappa, \tau_1, 2) + \mathrm{Adv}^{\mathsf{ind-cpa}}_{\Pi}(\kappa, \tau_2, 2) + \mathrm{Adv}^{\mathsf{ind-cpa}}_{\mathcal{E}}(\kappa, \tau_3).$$

We prove the lemma in the full version [PPS12].

It follows immediately from the lemma that as long as the message is complete, i. e. no blocks were dropped, any collusion of up to t users can be traced.

We now consider the case where the adversary is allowed to drop blocks. If there are no collusions, the case is unproblematic if robust codes are used (that resist erasures), because if the adversary drops a fraction δ of the blocks, he cannot erase significantly more than a fraction δ of the bits in the codeword.

Theorem 7. *Even if an adversary with a single user secret key can drop a fraction η of the message, if the 2ABE scheme Π is both IND-CPA and ANO-CPA, the symmetric encryption scheme $\mathcal{E}$ is IND-CPA, and the code $\mathcal{T}$ is t-fingerprinting for $\mathcal{FS}^\star$ for a fraction $\delta > \eta$ of erasures, then our construction Ψ is traceable. More precisely, one needs*

$$(\delta - \eta)^2 \geq \frac{1}{2} \times \mathrm{Adv}_{\Psi}^{\mathsf{find}}(\kappa, \tau, N, t, \varepsilon).$$

The proof is in the full version [PPS12]. We now show that our improvement from Section 3.3 allows tracing collusions of users even if they can drop blocks.

Theorem 8. *Even if an adversary with a single user secret key can drop a fraction η of the message, if the 2ABE scheme Π is both IND-CPA and ANO-CPA, the symmetric encryption scheme $\mathcal{E}$ is IND-CPA, and the code $\mathcal{T}$ is t-fingerprinting with error $\varepsilon/2$, then our construction Ψ with ρ repetitions of each bit of the codeword and $n\rho$ dummy bits is t-fingerprinting with error at most ε as long as*

$$(2\eta)^\rho \leq \frac{\varepsilon}{2n}.$$

4.3 A 2-user Anonymous Broadcast Encryption Scheme

We now present a concrete instance of a 2ABE scheme to use as a building block in our message-based traitor tracing scheme. We view the 2-key 1-copyrighted public-key encryption scheme of Kiayias and Yung [KY02], as a 2-user 1-collusion-secure anonymous broadcast encryption scheme (2ABE). For ease of exposition, we model the scheme as a KEM.

Let G be a group of prime order q, with a generator g. The public parameters consist of (G, q, g). Since we consider the 2-user case, we drop the N parameter:

- $\mathsf{Setup}(1^\kappa)$ picks $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q^\times$. For the two user-keys one chooses $d'_0, d'_1 \in \mathbb{Z}_q$, and sets $\mathsf{usk}_u \stackrel{\mathsf{def}}{=} (d_u = \alpha - d'_u \cdot \beta, d'_u)$, for $u = 0, 1$. The encryption key is $\mathsf{ek} \stackrel{\mathsf{def}}{=} \{(f = g^\alpha, h = g^\beta), \mathsf{upk}_0 = h^{d'_0}, \mathsf{upk}_1 = h^{d'_1}\}$.
- $\mathsf{Encaps}(\mathsf{ek}, S; r)$ where $r \in \mathbb{Z}_q^\times$
  - if $S = \{0, 1\}$ then $c = (g^r, h^r), K = f^r$
  - else if $S = \{u\}$ then $r' \xleftarrow{\$} \mathbb{Z}_q^\times$, and $c = (g^r, h^{r'}), K = (f/\mathsf{upk}_u)^r \times \mathsf{upk}_u^{r'}$
- $\mathsf{Decaps}(\mathsf{usk}_u, c)$ computes $K = c_0^{d_u} c_1^{d'_u}$. This is equal to $g^{r d_u} \times h^{r' d'_u} = (f/\mathsf{upk}_u)^r \times \mathsf{upk}_u^{r'}$, which ensures correctness in the second case, where $S = \{u\}$. In the first case, since $r' = r$, both users get the same key f^r.

Table 1. Comparison: G is the bit-length of a group element; B is the bit-length of a message block. |PTXT| is always $\ell n\mathsf{B}$, except for the schemes from Section 3.3 and 4, where it is $2\ell n\rho\mathsf{B}$

Construction in	$\lvert\mathsf{EK}\rvert$	$\lvert\mathsf{USK}\rvert$	|KeyHeader|	|CTXT|	Ciphertext Rate
Section 3.1	$2\ell n\mathsf{G}$	$\ell n\mathsf{G}$	$\ell n\mathsf{G}$	$2\ell n\mathsf{B}$	$2+\mathsf{G}/\mathsf{B}$
Section 3.1 + rr.	$2\ell n\mathsf{G}$	$\ell n\mathsf{G}$	G	$2\ell n\mathsf{B}$	$2+1\mathsf{G}/(\ell n\mathsf{B})$
Section 3.2	$2(\ell+1)n\mathsf{G}$	$\ell n\mathsf{G}$	$(\ell+1)n\mathsf{G}$	$(\ell+1)n\mathsf{B}$	$1+1/\ell+(1+1/\ell)\mathsf{G}/\mathsf{B}$
Section 3.3	$4(\ell+1)n\rho\mathsf{G}$	$2\ell n\rho\mathsf{G}$	$2(\ell+1)n\rho\mathsf{G}$	$2(\ell+1)n\rho\mathsf{B}$	$1+1/\ell+(1+1/\ell)\mathsf{G}/\mathsf{B}$
Section 3.4	$2(\ell+1)n\mathsf{G}$	$\ell n\mathsf{G}$	G	$(\ell+1)n\mathsf{B}$	$1+1/\ell+1\mathsf{G}/(\ell n\mathsf{B})$
Section 4	$8n\rho\mathsf{G}$	$4n\rho\mathsf{G}$	$4(\ell+1)n\rho\mathsf{G}$	$2(\ell+1)n\rho\mathsf{B}$	$1+1/\ell+(2+2/\ell)\mathsf{G}/\mathsf{B}$

This is a broadcast encryption, because when $S=\{u\}$, the user $1-u$ decapsulates differently. Anonymity comes from the fact that a ciphertext is either a Diffie-Hellman pair, when $S=\{0,1\}$, and a random pair in the other case.

4.4 Security of the 2ABE

Theorem 9. *If solving the DDH problem in the underlying group is hard, then the 2ABE scheme presented in Section 4.3 is ANO-ACPA-secure and*

$$\mathrm{Adv}^{\mathsf{ano-acpa}}_{2ABE}(\kappa,\tau)\leq 4\cdot\mathrm{Adv}^{ddh}(\kappa,\tau+\tau').$$

The proof of this theorem is given in the full version [PPS12].

Theorem 10. *If solving the DDH problem in the underlying group is hard, then the 2ABE scheme presented in Section 4.3 is a 2-user IND-CPA-secure BE scheme and*

$$\mathrm{Adv}^{\mathsf{ind-cpa}}_{2ABE}(\kappa,\tau)\leq \mathrm{Adv}^{ddh}(\kappa,\tau+\tau').$$

The proof of this theorem is given in the full version [PPS12].

5 Conclusion

Table 1 shows a comparison of several ways to do message-traceable encryption. We compare the length of the encryption key ($\lvert\mathsf{EK}\rvert$) and the user secret key ($\lvert\mathsf{USK}\rvert$), and the length of the key header and the symmetric encryption of the plaintext (CTXT), normalizing for a plaintext (PTXT) length (before marking blocks) of ℓn blocks. The ciphertext rate is defined as (|KeyHeader| + |CTXT|) / |PTXT|.

The simplest way is to use any public-key encryption scheme to encrypt each message block twice, described in Section 3.1: two pairs of keys are generated for each message block. Using ElGamal, we have one group element for the key header (the key encapsulation) per message-version block. We can reduce the key header to one group element by reusing randomness. However, even with randomness reuse (rr), using this method it is impossible to reduce the ciphertext rate below 2 without leaving some part of the message unprotected and exposed

to untraceable rebroadcasting, since each message block is encrypted twice with the symmetric keys.

Using our improved construction from Section 3.2, we immediately cut the number of blocks that must be sent almost in half with only a small constant increase in the key header as compared to plain ElGamal. The modifications from Section 3.3 multiply key and ciphertext length by 2ρ, but does not change the ciphertext rate. Reusing the randomness in the ciphertext, described in Section 3.4, again shrinks the key header that must be transmitted with every message.

Our main construction from Section 4 achieves the same asymptotic efficiency as the PKE construction, but with keys that are shorter by a factor of ℓ.

Since these results are all asymptotic, the question arises how our scheme performs in practice. Assume we want to design a scheme for $N = 2^{25}$ users, with a maximum false positive rate of $\varepsilon = 2^{-40}$. We set the collusion threshold to $t = 16$, which means we assume that retrieving the keys from 16 different decoders is prohibitively expensive (16 is the largest collusion threshold considered in [JL07, Fig. 5]). Then the length of the code is $n = d_m t^2 \log(N/\varepsilon)$ for some constant d_m. Blayer and Tassa claim that in most real-world applications, one can find a code with $d_m < 8$ [BT08], giving us a code length of 133 120 bits in exchange for a higher false-negative error rate. We assume that we only want to protect against pirates that rebroadcast at least 97% of the blocks, and set $\eta = 2^{-5}$. Since we need $(2\eta)^\rho \leq \varepsilon/2n$, $\rho = 17$, and we double the code length by padding with dummy bits, then our modified code has a length of 4 526 080 bits. We choose the efficiency parameter $\ell = 20$, so we need to split the film into 90 521 600 blocks.

Assuming a block size of 1 kB, our film needs to be at least 90.5 GB.[2] Because of the doubled blocks, the symmetrically encrypted part would be about 95 GB, and the length of the key header would be about 3.8 GB (assuming 160 bit group elements on a suitable elliptic curve), for a total ciphertext size of 98.8 GB that would comfortably fit on an existing 100 GB BDXL Blu-ray disc. The overhead of 9.2% is already within the acceptable range ([JL07] state 10% as their limit), and as we can expect the size of media formats to grow, the concrete efficiency of our scheme will only increase.

Acknowledgments. This work was supported by the French ANR-09-VERS-016 BEST Project and the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II. The authors would like to thank the anonymous reviewers of the Latincrypt 2012 program committee for their helpful comments.

References

[ABN10] Abdalla, M., Bellare, M., Neven, G.: Robust Encryption. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 480–497. Springer, Heidelberg (2010)

[2] To relate our ciphertext length to existing storage media sizes, we use SI prefixes.

[BBKS07] Bellare, M., Boldyreva, A., Kurosawa, K., Staddon, J.: Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security. IEEE Trans. on Info. Theory 53(11), 3927–3943 (2007)

[BF99] Boneh, D., Franklin, M.K.: An Efficient Public Key Traitor Scheme (Extended Abstract). In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 338–353. Springer, Heidelberg (1999)

[BN08] Boneh, D., Naor, M.: Traitor tracing with constant size ciphertext. In: ACM CCS, pp. 455–470 (2008)

[BP08] Billet, O., Phan, D.H.: Efficient Traitor Tracing from Collusion Secure Codes. In: Safavi-Naini, R. (ed.) ICITS 2008. LNCS, vol. 5155, pp. 171–182. Springer, Heidelberg (2008)

[BP09] Billet, O., Phan, D.H.: Traitors Collaborating in Public: Pirates 2.0. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 189–205. Springer, Heidelberg (2009)

[BS98] Boneh, D., Shaw, J.: Collusion-secure fingerprinting for digital data. IEEE Trans. on Information Theory 44(5), 1897–1905 (1998)

[BT08] Blayer, O., Tassa, T.: Improved versions of tardos' fingerprinting scheme. Designs, Codes and Cryptography 48(1), 79–103 (2008)

[CFN94] Chor, B., Fiat, A., Naor, M.: Tracing Traitors. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 257–270. Springer, Heidelberg (1994)

[FNP07] Fazio, N., Nicolosi, A., Phan, D.H.: Traitor Tracing with Optimal Transmission Rate. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 71–88. Springer, Heidelberg (2007)

[FT99] Fiat, A., Tassa, T.: Dynamic Traitor Tracing. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 354–371. Springer, Heidelberg (1999)

[JL07] Jin, H., Lotspiech, J.: Renewable Traitor Tracing: A Trace-Revoke-Trace System For Anonymous Attack. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 563–577. Springer, Heidelberg (2007)

[KP09] Kiayias, A., Pehlivanoglu, S.: Tracing and Revoking Pirate Rebroadcasts. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 253–271. Springer, Heidelberg (2009)

[KP10] Kiayias, A., Pehlivanoglu, S.: Encryption for Digital Content. In: Advances in Information Security, vol. 52. Springer (2010)

[KY02] Kiayias, A., Yung, M.: Traitor Tracing with Constant Transmission Rate. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 450–465. Springer, Heidelberg (2002)

[LPQ11] Libert, B., Paterson, K.G., Quaglia, E.A.: Anonymous broadcast encryption. Cryptology ePrint Archive, Report 2011/476 (2011)

[NSS99] Naccache, D., Shamir, A., Stern, J.P.: How to Copyright a Function? In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 188–196. Springer, Heidelberg (1999)

[PPS11] Phan, D.H., Pointcheval, D., Strefler, M.: Security Notions for Broadcast Encryption. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 377–394. Springer, Heidelberg (2011)

[PPS12] Phan, D.H., Pointcheval, D., Strefler, M.: Message-based traitor tracing with optimal ciphertext rate. In: Latincrypt, full version available from the author's webpage (2012)

[Sir07] Sirvent, T.: Traitor tracing scheme with constant ciphertext rate against powerful pirates. In: Tillich, J.-P., Augot, D., Sendrier, N. (eds.) Proc. of Workshop on Coding and Cryptography (WCC 2007), pp. 379–388 (April 2007)

[SNW00] Safavi-Naini, R., Wang, Y.: Sequential Traitor Tracing. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 316–332. Springer, Heidelberg (2000)

[Tar08] Tardos, G.: Optimal probabilistic fingerprint codes. Journal of the ACM 55(2) (May 2008)

A Definitions

A.1 Public-Key Encryption

Definition 11 (Encryption Scheme). *A public-key encryption scheme is a 4-tuple of algorithms* $\mathcal{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$*:*

- $\mathsf{Setup}(1^k)$*, where* k *is the security parameter, generates the global parameters* param *of the system;*
- $\mathsf{KeyGen}(\mathsf{param})$ *generates a pair of keys, the public (encryption) key* ek *and the associated private (decryption) key* dk*;*
- $\mathsf{Encrypt}(\mathsf{ek}, m; r)$ *produces a ciphertext* c *on the input message* m *and the public key* ek*, using the random coins* r *(we may omit* r *when the notation is obvious);*
- $\mathsf{Decrypt}(\mathsf{dk}, c)$ *decrypts the ciphertext* c *under the private key* dk*. It outputs the plaintext, or* $\perp$ *if the ciphertext is invalid.*

The correctness requirement is that we get $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{Encrypt}(\mathsf{ek}, m)) = m$ if $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ for all parameters.

Such an encryption scheme is said to be (t, q_D, ν)-IND-CCA-secure (semantic security against chosen-ciphertext attacks) if in the security game presented in Figure 4, the advantage, denoted $\mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k, t, q_D)$, of any t-time adversary $\mathcal{A}$ asking at most q_D decryption queries to the $\mathsf{ODecrypt}$ oracle is bounded by ν:

$$\mathrm{Adv}^{\mathsf{ind-cca}}_{\mathcal{PKE}}(k, t, q_D) = \max_{\mathcal{A}} \{\Pr[\mathrm{Exp}^{\mathsf{ind-cca}-1}_{\mathcal{PKE},\mathcal{A}}(k) = 1] - \Pr[\mathrm{Exp}^{\mathsf{ind-cca}-0}_{\mathcal{PKE},\mathcal{A}}(k) = 1]\}.$$

This definition includes IND-CPA (for Chosen-Plaintext Attacks) when $q_D = 0$.

A.2 Anonymous Broadcast Encryption

Anonymous broadcast encryption (ABE) allows to address a message to a subset of the users, without revealing this target set even to users who successfully decrypted the message. We define an ABE as a key encapsulation mechanism (KEM), following the definitions found in [PPS11, LPQ11]:

$\mathrm{Exp}^{\mathsf{ind-cca}-b}_{\mathcal{PKE},\mathcal{A}}(k)$
$\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$;
$\mathcal{Q}_D \leftarrow \emptyset$, $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$;
$(state, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{ODecrypt}(\cdot)}(\mathsf{FIND}; \mathsf{param}, \mathsf{ek})$;
$c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, m_b)$;
$b' \leftarrow \mathcal{A}^{\mathsf{ODecrypt}}(\mathsf{GUESS}, state; c^*)$;
`if` $c^* \in \mathcal{Q}_D$ `then return` 0;
`else return` b';

$\mathsf{ODecrypt}(c)$
$\mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{c\}$;
$m \leftarrow \mathsf{Decrypt}(\mathsf{dk}, c)$;
`return` m;

Fig. 4. $\mathcal{PKE}$: Semantic Security against Chosen-Ciphertext Attacks (IND-CCA)

- $\mathsf{Setup}(1^\kappa, N)$, where k is the security parameter, and N the number of users, generates the global parameters param of the system (omitted in the following), N user secret keys $\{\mathsf{usk}_i\}_{i=1,\ldots,N}$, and an encryption key ek.
- $\mathsf{Encaps}(\mathsf{ek}, S; r)$ takes as input the encryption key ek, the target set $S \subset \{1, \ldots, N\}$, and some random coins r (which are sometimes omitted). It outputs a session key K, and an encapsulation c of K;
- $\mathsf{Decaps}(\mathsf{usk}_i, c)$ takes as input a decryption key and a ciphertext c. It outputs the session key K, or the error symbol $\perp$.

For correctness, we require that for any c that encapsulates a key K for a target set S, if $i \in S$, then $\mathsf{Decaps}(\mathsf{usk}_i, c)$ outputs K. Then, semantic security and anonymity should be satisfied.

Definition 12 (Semantic security). *We say that an anonymous broadcast encryption (ABE) scheme Π is (τ, N, q_C, q_D, ν)-IND-ACCA-secure (semantic security against adaptive corruption and chosen-ciphertext attacks) if in the security game presented in Figure 5, the advantage, denoted* $\mathrm{Adv}^{\mathsf{ind-acca}}_{\Pi}(\kappa, \tau, N, q_C, q_D)$*, of any τ-time adversary $\mathcal{A}$ corrupting at most q_C users (*$\mathsf{OCorrupt}$ *oracle), and asking for at most q_D decryption queries (*$\mathsf{ODecrypt}$ *oracle), is bounded by ν:*

$$\mathrm{Adv}^{\mathsf{ind-acca}}_{\Pi}(\kappa, \tau, N, q_C, q_D) = \max_{\mathcal{A}}\{\Pr[\mathrm{Exp}^{\mathsf{ind-acca}-1}_{\Pi,\mathcal{A}}(\kappa, N) = 1] - \Pr[\mathrm{Exp}^{\mathsf{ind-acca}-0}_{\Pi,\mathcal{A}}(\kappa, N) = 1]\}.$$

This definition includes IND-ACPA (for Chosen-Plaintext Attacks) when $q_D = 0$, and thus we denote the advantage $\mathrm{Adv}^{\mathsf{ind-acpa}}_{\Pi}(\kappa, \tau, N, q_C)$. When no corruption is allowed, we denote the advantage $\mathrm{Adv}^{\mathsf{ind-cpa}}_{\Pi}(\kappa, \tau, N)$.

Definition 13 (Anonymity). *We say that an anonymous broadcast encryption (ABE) scheme Π is (τ, N, q_C, q_D, ν)-ANO-ACCA-secure (anonymity against adaptive corruption and chosen-ciphertext attacks) if in the security game presented in Figure 5, the advantage, denoted* $\mathrm{Adv}^{\mathsf{ano-acca}}_{\Pi}(\kappa, \tau, N, q_C, q_D)$*, of any τ-time adversary $\mathcal{A}$ corrupting at most q_C users (*$\mathsf{OCorrupt}$ *oracle), and asking for at most q_D decryption queries (*$\mathsf{ODecrypt}$ *oracle), is bounded by ν:*

$\mathsf{ODecrypt}(i, c)$
$\quad \mathcal{Q}_D \leftarrow \mathcal{Q}_D \cup \{(i, c)\};$
$\quad K \leftarrow \mathsf{Decaps}(\mathsf{usk}_i, c);$ `return` $K;$

$\mathsf{OCorrupt}(i)$
$\quad \mathcal{Q}_C \leftarrow \mathcal{Q}_C \cup \{i\};$
$\quad$ `return` $\mathsf{usk}_i;$

$\mathrm{Exp}^{\mathsf{ind-acca}-b}_{\Pi,\mathcal{A}}(\kappa, N)$
$\quad (\{\mathsf{usk}_i\}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\kappa, N);\ \mathcal{Q}_C \leftarrow \emptyset;\ \mathcal{Q}_D \leftarrow \emptyset;$
$\quad (state, S) \leftarrow \mathcal{A}^{\mathsf{ODecrypt}(\cdot,\cdot),\mathsf{OCorrupt}(\cdot)}(\mathsf{FIND}; \mathsf{ek});$
$\quad (K_1, c^*) \leftarrow \mathsf{Encaps}(\mathsf{ek}, S);\ K_0 \xleftarrow{\$} \mathcal{K}$
$\quad b' \leftarrow \mathcal{A}^{\mathsf{ODecrypt}(\cdot,\cdot),\mathsf{OCorrupt}(\cdot)}(\mathsf{GUESS}; state, K_b, c^*);$
$\quad$ `if` $\exists i \in S : (i, c^*) \in \mathcal{Q}_D$ `or` $S \cap \mathcal{Q}_C \neq \emptyset$ `then return` $0;$
$\quad$ `else return` $b';$

$\mathrm{Exp}^{\mathsf{ano-acca}-b}_{\Pi,\mathcal{A}}(\kappa, N)$
$\quad (\{\mathsf{usk}_i\}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\kappa, N);\ \mathcal{Q}_C \leftarrow \emptyset;\ \mathcal{Q}_D \leftarrow \emptyset;$
$\quad (state, S_0, S_1) \leftarrow \mathcal{A}^{\mathsf{ODecrypt}(\cdot,\cdot),\mathsf{OCorrupt}(\cdot)}(\mathsf{FIND}; \mathsf{ek});$
$\quad (K, c^*) \leftarrow \mathsf{Encaps}(\mathsf{ek}, S_b);$
$\quad b' \leftarrow \mathcal{A}^{\mathsf{ODecrypt}(\cdot,\cdot),\mathsf{OCorrupt}(\cdot)}(\mathsf{GUESS}, state; K, c^*);$
$\quad$ `if` $\exists i \in \mathcal{S}_0 \triangle S_1 : (i, c^*) \in \mathcal{Q}_D$ `or` $(\mathcal{S}_0 \triangle S_1) \cap \mathcal{Q}_C \neq \emptyset$
$\quad$ `then return` $0;$ `else return` $b';$

Fig. 5. Security games for ABE

$$\mathrm{Adv}^{\mathsf{ano-acca}}_{\Pi}(\kappa, \tau, N, q_C, q_D) = \max_{\mathcal{A}}\{\Pr[\mathrm{Exp}^{\mathsf{ano-acca}-1}_{\Pi,\mathcal{A}}(\kappa, N) = 1] - \Pr[\mathrm{Exp}^{\mathsf{ano-acca}-0}_{\Pi,\mathcal{A}}(\kappa, N) = 1]\}.$$

This definition includes ANO-ACPA (for Chosen-Plaintext Attacks) when $q_D = 0$, and thus we denote the advantage $\mathrm{Adv}^{\mathsf{ano-acpa}}_{\Pi}(k, \tau, N, q_C)$. When no corruption is allowed, we denote the advantage $\mathrm{Adv}^{\mathsf{ano-cpa}}_{\Pi}(\kappa, \tau, N)$.

Leakage-Resilient Spatial Encryption

Michel Abdalla and Jill-Jênn Vie

Département d'Informatique, École Normale Supérieure, France
{Michel.Abdalla,Jill-Jenn.Vie}@ens.fr
http://www.di.ens.fr/users/{mabdalla,jvie}

Abstract. Spatial encryption is a generic public-key cryptosystem where vectors play the role of public keys and secret keys are associated to affine spaces. Any secret key associated to a space can decrypt all ciphertexts encrypted for vectors in that space, and the delegation relation is defined by subspace inclusion. Though several constructions of spatial encryption schemes have been proposed in the literature, none of them are known to remain secure in the leakage-resilient setting, in which the adversary may be capable of learning limited additional information about the master secret key and other secret keys in the system. In this paper, we propose the first spatial encryption scheme achieving leakage resilience in the standard model, based on existing static assumptions over bilinear groups of composite order. Our new scheme is based on the leakage-resilient HIBE scheme by Lewko, Rouselakis, and Waters in TCC 2011 and can be seen as a generalization of Moriyama-Doi spatial encryption scheme to the leakage-resilient setting.

Keywords: Spatial encryption, leakage resilience.

1 Introduction

When a theoretical scientist designs a cryptosystem, he is often unpleasantly surprised by the weaknesses that its physical implementation can induce. Indeed, history has proven that these attempts to break the system, called *side-channel attacks*, can be devastating. In order to design schemes that can provably resist these attacks, a stronger notion of security has been established. A *leakage-resilient* scheme considers that an attacker is able to learn partial information about some secret values used throughout the lifetime of the system.

1.1 Leakage Models

This paper will be mainly concerned with achieving leakage resilience based on the model of *memory attacks*, introduced by Akavia, Goldwasser, and Vaikuntanathan in [AGV09]. According to their model, the adversary can learn *arbitrary* information about the secret state of a system, by selecting polynomial-time computable functions $f_i : \{0,1\}^* \to \{0,1\}^{\lambda_i}$ and learning the value of f_i applied to the internal state of the system.

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 78–99, 2012.

As mentioned in [BSW11], there are two main variants of the memory attacks model, known as the bounded-leakage and the continual-leakage models.

In a *bounded-leakage* model, we define a *leakage bound* λ such that the overall amount of information learned by the adversary throughout the entire lifetime of the system is $\sum_i \lambda_i \leq \lambda$.

If a system is used continually for a sufficiently long time, then any static piece of information that stays unmodified on the system can eventually be recovered by the adversary. Thus, the secret keys of such systems must be periodically *refreshed*. Recently, Dodis et al. [DHLAW10] and Brakerski et al. [BKKV10] concurrently suggested the *continual-leakage* model, which allows for its internal secret keys to be refreshed periodically without changing the value of the corresponding public key. In their model, the adversary is allowed to learn functions of the entire secret state of the system all the time, including the secret key itself and the random bits generated by the system. Moreover, the leakage bound λ only refers to the overall amount of information learned by the adversary between any two successive refreshes. Hence, we can think of λ as the *leakage rate* of the scheme.

1.2 Spatial Encryption

The goal of this work is to consider leakage resilience in the context of spatial encryption, which is a generic public-key cryptosystem first defined in [BH08] where vectors play the role of public keys and secret keys are associated to affine spaces. In a spatial encryption scheme, any user possessing a secret key for a certain space W_1 can *delegate* a secret key for a space W_2 included in W_1. A message encrypted for a certain vector $\boldsymbol{u}$ can be decrypted by any user possessing a secret key associated to a space that contains $\boldsymbol{u}$.

In addition to proposing the concept of spatial encryption, Boneh and Hamburg [BH08] also provided a construction based on bilinear maps which is provably secure in the selective security model based on the BDDHE assumption ([BGW05]). In the selective secure model, the adversary decides on the vector upon which he wants to be challenged before seeing the public key of the scheme. Their work was later improved by Moriyama and Doi in [MD11] who proposed a fully secure (as opposed to selective secure) spatial encryption scheme in the standard model based on bilinear groups of composite order.

1.3 Our Contributions

In this paper, we propose the first spatial encryption scheme achieving leakage resilience in the bounded leakage model. Our scheme is based on the leakage-resilient HIBE scheme by Lewko, Rouselakis, and Waters in [LRW11] and can be seen as a generalization of Moriyama-Doi spatial encryption scheme to the leakage-resilient setting. Like the Lewko-Rouselakis-Waters HIBE scheme, our new scheme works over bilinear groups of composite order and is proven secure under static assumptions used in [LOS+10, LW10, LRW11].

1.4 Outline

In Section 2, we recall some basic definitions and state the complexity assumptions that will be needed for the proof of security of our scheme. In Section 2.4, we recall the formal definition of spatial encryption, which is a generalization of hierarchical identity-based encryption. In Section 3, we present our leakage-resilient spatial encryption scheme. It is worth noting that, to date, none of the existing constructions of spatial encryption are leakage resilient and that our scheme is the first one to achieve this level of security. We conclude this article by listing some possible directions for future work.

2 Preliminaries

2.1 Notation and Conventions

We denote by $a, b, c \xleftarrow{R} S$ the fact that a, b, c are picked independently and uniformly at random from a set S. By $\mathsf{negl}(\lambda)$, we denote a negligible function of λ, i.e. a function $f : \mathbb{N} \to \mathbb{R}$ such that for every $c > 0$ and for all but a finite number of λ's: $f(\lambda) \leq \lambda^{-c}$.

By $|x|$, we denote the size/number of bits of term x. The empty vector is denoted ε. Also, the special symbol $\perp$ is meant to serve as a unique dummy value in all our systems. Finally, by PPT we denote a probabilistic polynomial-time algorithm.

2.2 Composite Order Bilinear Groups

Composite order bilinear groups were first used in cryptography in [BGN05] (see also [Bon07]). To generate these groups, we suppose the existence of an efficient group generator algorithm G which takes as input the security parameter λ and outputs a description $\mathbb{I} = (N, \mathbb{G}, \mathbb{G}_T, \hat{e})$ of a bilinear setting, where $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order N, and $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a map with the following properties:

1. (bilinearity) $\forall g, h \in \mathbb{G}$ and $a, b \in \mathbb{Z}_N$ it holds that $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$;
2. (non-degeneracy) $\exists g \in \mathbb{G}$ such that $\hat{e}(g, g)$ has order N in $\mathbb{G}_T$.

We assume that the group descriptions of $\mathbb{G}$ and $\mathbb{G}_T$ include generators of the respective cyclic groups. We require that the group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map $\hat{e}$ are computable in deterministic polynomial time in λ. In our construction, we will make hardness assumptions for bilinear settings whose order N is product of three distinct primes each of length $\Theta(\lambda)$. For an integer m dividing N, we let $\mathbb{G}_m$ denote the subgroup of $\mathbb{G}$ of order m. From the fact that the group is cyclic, it is easy to verify that if g and h are group elements of co-prime orders then $\hat{e}(g, h) = 1$. This is called the *orthogonality property* and is a crucial tool in our constructions.

2.3 Complexity Assumptions

The proof of security of our system will rely on Assumptions 1, 2, and 3 from [LRW11]. These assumptions work over bilinear groups of composite order and were already used in the security proof of several other systems (e.g., [LOS+10, LW10]). For completeness, we restate these assumptions below.

To simplify, we will write in the proof $T \in \mathbb{G}_{p_1 p_2^? p_3}$ when T is a challenge term either in $\mathbb{G}_{p_1 p_3}$ or in $\mathbb{G}_{p_1 p_2 p_3}$, for example.

Assumption 1. Given $\mathcal{D}_1 = (N, \mathbb{G}, \mathbb{G}_T, \hat{e}, g_1, g_3)$, no PPT (probabilistic polynomial-time) adversary has a non-negligible advantage in distinguishing

$$T_1 = g_1^a \text{ from } T_2 = g_1^a g_2^b$$

where $a, b \xleftarrow{R} \mathbb{Z}_N$.

The advantage of an algorithm $\mathcal{A}$ in breaking Assumption 1 is defined as:

$$\mathsf{Adv}_1^{\mathcal{A}}(\lambda) = \left|\Pr[\mathcal{A}(\mathcal{D}_1, T_1) = 1] - \Pr[\mathcal{A}(\mathcal{D}_1, T_2) = 1]\right|$$

We say that Assumption 1 holds if for all PPT $\mathcal{A}$, $\mathsf{Adv}_1^{\mathcal{A}}(\lambda)$ is a negligible function of λ.

Assumption 2. Given $\mathcal{D}_2 = (N, \mathbb{G}, \mathbb{G}_T, \hat{e}, g_1, g_3, g_1^x g_2^y, g_2^{y'} g_3^{z'})$ where $x, y, y', z' \xleftarrow{R} \mathbb{Z}_N$, no PPT adversary has a non-negligible advantage in distinguishing

$$T_1 = g_1^a g_3^c \text{ from } T_2 = g_1^a g_2^b g_3^c$$

where $a, b, c \xleftarrow{R} \mathbb{Z}_N$.

The advantage of an algorithm $\mathcal{A}$ in breaking Assumption 2 is defined as:

$$\mathsf{Adv}_2^{\mathcal{A}}(\lambda) = \left|\Pr[\mathcal{A}(\mathcal{D}_2, T_1) = 1] - \Pr[\mathcal{A}(\mathcal{D}_2, T_2) = 1]\right|$$

We say that Assumption 2 holds if for all PPT $\mathcal{A}$, $\mathsf{Adv}_2^{\mathcal{A}}(\lambda)$ is a negligible function of λ.

Assumption 3. Given $\mathcal{D}_3 = (N, \mathbb{G}, \mathbb{G}_T, \hat{e}, g_1, g_2, g_3, g_1^x g_2^y, g_1^{x'} g_2^{y'})$ where $x, y, x', y' \xleftarrow{R} \mathbb{Z}_N$, no PPT adversary has a non-negligible advantage in distinguishing

$$T_1 = \hat{e}(g_1, g_1)^{xx'} \in \mathbb{G}_T \text{ from } T_2 \xleftarrow{R} \mathbb{G}_T.$$

The advantage of an algorithm $\mathcal{A}$ in breaking Assumption 3 is defined as:

$$\mathsf{Adv}_3^{\mathcal{A}}(\lambda) = \left|\Pr[\mathcal{A}(\mathcal{D}_3, T_1) = 1] - \Pr[\mathcal{A}(\mathcal{D}_3, T_2) = 1]\right|$$

We say that Assumption 3 holds if for all PPT $\mathcal{A}$, $\mathsf{Adv}_3^{\mathcal{A}}(\lambda)$ is a negligible function of λ.

2.4 Spatial Encryption

The notion of spatial encryption proposed by Boneh and Hamburg in [BH08] is a generalization of the notion of hierarchical identity-based encryption [HL02], in which vectors play the role of public keys and secret keys are associated to affine spaces. In these schemes, any user possessing a secret key for a certain space W_1 can *delegate* a secret key for a space W_2 included in W_1. A message encrypted for a certain vector $\boldsymbol{u}$ can be decrypted by any user possessing a secret key associated to a space that contains $\boldsymbol{u}$. We denote $\mathrm{Aff}(M, \boldsymbol{u})$ the affine space $\{\boldsymbol{u} + M\boldsymbol{v}, \boldsymbol{v} \in \mathbb{Z}_N^n\}$. All affine spaces considered will be subspaces of $E = \mathbb{Z}_N^n$, the whole space.

More formally, a spatial encryption scheme is defined by the following set of algorithms.

$\mathsf{Setup}(1^\lambda, n) \to (\mathrm{PP}, \mathrm{SK}_E)$ The setup algorithm takes an integer security parameter λ as input and outputs the public parameters PP and the original key for the whole space SK_E. In the rest of this paper, all algorithms will take implicitly the security parameter and the public parameters as inputs.

$\mathsf{KeyGen}(\mathrm{SK}_E, W) \to K$ The key generation algorithm takes a key for the whole space SK_E and an affine space W. It outputs a secret key $K = \mathrm{SK}_W$, for the affine space W. If $W = E$, it outputs another key for the whole space, $K = \mathrm{SK}'_E$, such that $|\mathrm{SK}_E| = |\mathrm{SK}'_E|$. This new key can now be used instead of the original key in calls of KeyGen.

$\mathsf{Encrypt}(M, \boldsymbol{u}) \to \mathrm{CT}$ The encryption algorithm takes in a message M and a vector $\boldsymbol{u}$. It outputs a ciphertext CT.

$\mathsf{Delegate}(W_1, \mathrm{SK}_{W_1}, W_2) \to \mathrm{SK}_{W_2}$ The algorithm takes in an affine space W_1, a secret key for that space, and a subspace W_2 included in W_1. It outputs a secret key for W_2.

$\mathsf{Decrypt}(\mathrm{CT}, \boldsymbol{u}, W, \mathrm{SK}_W) \to m$ The decryption algorithm takes in a ciphertext CT for message m and for vector $\boldsymbol{u}$, and a secret key SK for space W. If W contains $\boldsymbol{u}$, it first delegates SK_W to obtain $SK_{\boldsymbol{u}}$. Then it outputs the message m.

Security. The standard notion of security for spatial encryption schemes is indistinguishability of plaintexts under chosen-plaintext attacks [BH08], in which the adversary cannot distinguish between the encryption of two equal-length messages of its choice for a challenge vector, even if it is given access to secret keys corresponding to affine spaces that do not contain the challenge vector. In this paper, we propose an extension of the above security notion, which also takes into account the leakage of the secret keys used throughout the lifetime of the system.

The new notion of leakage-resilient security for spatial encryption schemes is based on a game, called $\mathsf{MasterLeakSpatial}$, which is an adaptation of the notion

of master-key leakage security (MasterLeak) for HIBE schemes introduced by Lewko, Rouselakis and Waters in [LRW11] to the spatial encryption setting. In this game, the challenger first makes a call to Setup to get a secret key for the whole space, and public parameters that it can give to the adversary, which runs in two phases. In a first phase, the adversary can make a polynomial number of **Create**, **Delegate**, **Leak**, and **Reveal** queries, in any order.

- Via a **Create** query, the adversary can ask the challenger to create a key for a space W and to store it in an array. The adversary has to additionally supply a handle h which indicates the master key to be used during the key generation. In return, the challenger outputs a unique handle associated with the new key or $\perp$ if the handle provided by the adversary does not correspond to a master key.
- Via a **Delegate** query, the adversary can ask the challenger to create a key for a space W' starting from a key associated with a handle h. Let W and SK_W be the space and the corresponding key associated with the handle h. For simplicity, we assume that W contains W'. To answer this query, the challenger computes the new key by running the Delegate algorithm, stores it in a new position of the array, and returns the unique handle associated with it to the adversary.
- Via a **Leak** query, the adversary can ask the challenger to compute a leak function f to the key associated with a handle h. Let $W, \mathrm{SK}_W, \texttt{leaked}$ be the space, key, and total amount of leaked bits associated with the handle h. If $\texttt{leaked} + |f(\mathrm{SK})| \leq \ell_{\mathrm{SK}}$, then the challenger simply returns $f(\mathrm{SK})$ to the adversary. Otherwise, the challenger returns $\perp$.
- Via a **Reveal** query, the adversary can ask the challenger to recover a entire key associated with handle h, as long as it is not a key associated to the entire space.

At the end of the first phase, the adversary chooses a challenge vector $\boldsymbol{u}^*$ that should not be contained in a space associated with any of the revealed keys, as well as two challenge messages m_0, m_1. The adversary then gives $(\boldsymbol{u}^*, m_0, m_1)$ to the challenger, which in turn chooses a random bit β and returns $\mathrm{CT}^* = \mathsf{Encrypt}(m_\beta, \boldsymbol{u}^*)$ to the adversary. From then on, the second phase starts, and the adversary can make additional **Create** and **Delegate** queries. It can also make additional **Reveal** queries as well as long as the associated space does not contain $\boldsymbol{u}^*$. At the end of the second phase, the adversary outputs a guess value β' for β and wins if $\beta' = \beta$. Finally, the scheme is considered leakage resilient if no PPT adversary can win the MasterLeakSpatial game with probability significantly better than $1/2$.

In order to define more formally the MasterLeakSpatial security game, let us first introduce the following parameters:

- the set $\mathcal{R} \in \mathcal{P}(E)$ of spaces for which a key has been revealed;
- the array $\mathcal{T} \subset E \times \mathcal{SK} \times \mathbb{N}$ that holds tuples of the form $(\texttt{space}, \texttt{key}, \texttt{leaked})$.

The formal definition of the game MasterLeakSpatial, which is described in Figure 1, consists of the following phases:

Setup. The challenger makes a call to $\mathsf{Setup}(1^\lambda)$ and gets a key SK for the whole space E and the public parameters PP. It gives PP to the adversary.

Phase 1. In this phase, the adversary can make any of the following queries to the challenger, in any possible way: **Create**, **Leak**, **Reveal**, **Delegate**. These algorithms are listed in Figure 1.

Challenge. The adversary chooses a challenge vector $\boldsymbol{u}^*$ such that no element of $\mathcal{R}$ contains it, as well as two messages m_0, m_1 of equal size. It makes a call to $\mathbf{LR}(\boldsymbol{u}^*, m_0, m_1)$ and obtains a ciphertext CT^*.

Phase 2. This is the same as **Phase 1**, except that only queries to **Create**, **Delegate**, and **Reveal** are allowed. Moreover, **Reveal** should not refer to a secret key whose associated space contains $\boldsymbol{u}^*$.

Guess. The adversary chooses a bit β' and calls **Finalize**(β'). If the output is **True**, it succeeds.

Game MasterLeakSpatial

procedure Initialize
$(\mathrm{PP}, \mathrm{SK}) \xleftarrow{R} \mathsf{Setup}$
$\mathcal{R} \leftarrow \emptyset$
$\mathcal{T}[0] \leftarrow (E, \mathrm{SK}, 0)$
$\mathtt{nbKeys} \leftarrow 1$
$\beta \xleftarrow{R} \{0,1\}$
Return PP

procedure Create(h, W)
If $\mathcal{T}[h].\mathtt{space} = E$ then
  $\mathrm{SK}_W \xleftarrow{R} \mathsf{KeyGen}(\mathcal{T}[h].\mathtt{key}, W)$
  $h' \leftarrow \mathtt{nbKeys}$
  $\mathcal{T}[h'] \leftarrow (W, \mathrm{SK}_W, 0)$
  $\mathtt{nbKeys} \leftarrow \mathtt{nbKeys} + 1$
  Return h'
Else
  Return $\perp$

procedure Delegate(h, W')
$\mathrm{SK}_{W'} \leftarrow \mathsf{Delegate}(\mathcal{T}[h].\mathtt{space}, \mathcal{T}[h].\mathtt{key}, W')$
$h' \leftarrow \mathtt{nbKeys}$
$\mathcal{T}[h'] \leftarrow (W', \mathrm{SK}_{W'}, 0)$
$\mathtt{nbKeys} \leftarrow \mathtt{nbKeys} + 1$
Return h'

procedure Leak(h, f)
$\mathrm{SK} \leftarrow \mathcal{T}[h].\mathtt{key}$
If $\mathcal{T}[h].\mathtt{leaked} + |f(\mathrm{SK})| \leq \ell_{\mathrm{SK}}$ then
  $\mathcal{T}[h].\mathtt{leaked} \leftarrow \mathcal{T}[h].\mathtt{leaked} + |f(\mathrm{SK})|$
  Return $f(\mathrm{SK})$
Else
  Return $\perp$

procedure Reveal(h)
If $\mathcal{T}[h].\mathtt{space} \neq E$ then
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{T}[h].\mathtt{space}$
  Return $\mathcal{T}[h].\mathtt{key}$
Else
  Return $\perp$

procedure LR$(\boldsymbol{u}^*, m_0, m_1)$
$\mathrm{CT}^* \xleftarrow{R} \mathsf{Encrypt}(\boldsymbol{u}^*, m_\beta)$
Return CT^*

procedure Finalize(β')
Return $(\beta' = \beta)$

Fig. 1. Algorithms of the game MasterLeakSpatial

Finally, the security definition that we use is the following:

Definition 1. *A spatial encryption system Π is ℓ_{SK}-master-leakage secure if for all PPT adversaries $\mathcal{A}$ it is true that*

$$\mathsf{Adv}^{\mathsf{MasterLeakSpatial}}_{\mathcal{A},\Pi}(\lambda, \ell_{\mathrm{SK}}) \leq \mathsf{negl}(\lambda)$$

where $\mathsf{Adv}^{\mathsf{MasterLeakSpatial}}_{\mathcal{A},\Pi}(\lambda, \ell_{\mathrm{SK}})$ *is the advantage of* $\mathcal{A}$ *in game* MasterLeakSpatial *with security parameter* λ *and leakage parameter* $\ell_{\mathrm{SK}} = \ell_{\mathrm{SK}}(\lambda)$ *and is formally defined as:*

$$\mathsf{Adv}^{\mathsf{MasterLeakSpatial}}_{\mathcal{A},\Pi}(\lambda, \ell_{\mathrm{SK}}) = \left| \Pr[\mathcal{A} \text{ succeeds}] - \frac{1}{2} \right|,$$

where the probability is over all random bits used by the challenger and the attacker.

3 Our New Leakage-Resilient Spatial Encryption Scheme

We now present our construction of a leakage-resilient spatial encryption scheme, which can be seen as an extension of the leakage-resilient HIBE scheme by Lewko, Rouselakis, and Waters in [LRW11]. As in the original scheme, ciphertexts contain a constant number of group elements. The security proof of this system is available in Appendix A.

3.1 A Few Useful Notations

For a vector $v = (v_1, \ldots, v_n)^\top \in \mathbb{Z}_N^n$ of field elements, we use $g^{\boldsymbol{v}}$ to denote the vector of group elements

$$g^{\boldsymbol{v}} = (g^{v_1}, \ldots, g^{v_n})^\top \in \mathbb{G}^n$$

To simplify the notation, we introduce an operator

$$\begin{aligned} \psi : \mathbb{G}^n \times \mathbb{Z}_N^n &\to \mathbb{G} \\ (g^{\boldsymbol{v}}, w) &\mapsto g^{\langle \boldsymbol{v}, \boldsymbol{w} \rangle} \end{aligned}$$

and its extension:

$$\begin{aligned} \psi^* : \mathbb{G}^n \times \mathbb{Z}_N^{n \times d} &\to \mathbb{G}^d \\ \left(g^{\boldsymbol{v}}, \left(\boldsymbol{C_1}, \ldots, \boldsymbol{C_d} \right) \right) &\mapsto (\psi(g^{\boldsymbol{v}}, \boldsymbol{C_1}), \ldots, \psi(g^{\boldsymbol{v}}, \boldsymbol{C_d}))^\top \end{aligned}$$

We can notice that ψ is easily computable.

Given a pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, we can define an extension $\hat{e}_n$ in a similar way:

$$\begin{aligned} \hat{e}_n : \mathbb{G}^n \times \mathbb{G}^n &\to \mathbb{G}_T \\ ((g_1, \ldots, g_n), (g'_1, \ldots, g'_n)) &\mapsto \prod_{i=1}^{n} \hat{e}(g_i, g'_i) \end{aligned}$$

3.2 Construction

The system parameters for our spatial encryption system will be primes p_1, p_2, p_3 (where each $\log p_i$ is approximately the security parameter λ) and two groups $\mathbb{G}$ and $\mathbb{G}_T$ of order $N = p_1 p_2 p_3$, with a bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Additionally, the public parameters will include group elements $g_1 \in \mathbb{G}_{p_1}, g_3 \in \mathbb{G}_{p_3}, g_1^{\varphi} \in \mathbb{G}, t \in \mathbb{G}_T$ and a vector $g_1^{\boldsymbol{\alpha}} \in \mathbb{G}^n$.

A secret key for an affine space $W = \mathrm{Aff}(M, \boldsymbol{u})$ of dimension d will be composed of $d + n + 2$ elements from the group $\mathbb{G}$:

$$(\boldsymbol{k_\rho}, k_r, k_{\boldsymbol{u}}, \boldsymbol{k}_{\mathrm{del}}) = \left(g_1^{\boldsymbol{\rho}}, g_1^r, g_1^{\tau + r(\varphi + \langle \boldsymbol{u}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\rho}, \boldsymbol{\sigma} \rangle}, g_1^{rM^\top \boldsymbol{\alpha}}\right) \times g_3^{\boldsymbol{\mu}} \in \mathbb{G}^{d+n+2}.$$

$\mathsf{Setup}(\lambda)$ generates the system parameters $N = p_1 p_2 p_3, \mathbb{G}, \mathbb{G}_T$. It then chooses parameters

$$g_1 \xleftarrow{R} \mathbb{G}_{p_1} \qquad g_3 \xleftarrow{R} \mathbb{G}_{p_3} \qquad r, \varphi \xleftarrow{R} \mathbb{Z}_N \qquad \boldsymbol{\alpha}, \boldsymbol{\rho}, \boldsymbol{\sigma} \xleftarrow{R} \mathbb{Z}_N^n \qquad \boldsymbol{\mu} \xleftarrow{R} \mathbb{Z}_N^{2n+2}$$

and secret parameter $\tau \xleftarrow{R} \mathbb{Z}_N$, then computes $t = \hat{e}(g_1, g_1)^\tau$. It outputs public parameters

$$\mathrm{PP} = (N, g_1, g_3, g_1^{\varphi}, g_1^{\boldsymbol{\alpha}}, t, g_1^{\boldsymbol{\sigma}})$$

and secret key

$$\mathrm{SK}_E = \left(g_1^{\boldsymbol{\rho}}, g_1^r, g_1^{\tau + r\varphi - \langle \boldsymbol{\rho}, \boldsymbol{\sigma} \rangle}, g_1^{r\boldsymbol{\alpha}}\right) \times g_3^{\boldsymbol{\mu}}.$$

$\mathsf{Delegate}(W_1, \mathrm{SK}_{W_1}, W_2)$ takes two subspaces $W_1 = \mathrm{Aff}(M_1, \boldsymbol{u_1})$ and $W_2 = \mathrm{Aff}(M_2, \boldsymbol{u_2})$ and a key SK_{W_1} under the form $(\boldsymbol{k_\rho}, k_r, k_{\boldsymbol{u_1}}, \boldsymbol{k}_{\mathrm{del}})$. Let $\boldsymbol{\mu}$ be the $\mathbb{G}_{p_3}$ part of SK_{W_1}. We similarly divide $\boldsymbol{\mu}$ into $(\boldsymbol{\mu_\rho}, \mu_r, \mu_{\boldsymbol{u_1}}, \boldsymbol{\mu}_{\mathrm{del}})$. Since W_2 is a subspace of W_1, we must have $M_2 = M_1 T$ and $\boldsymbol{u_2} = \boldsymbol{u_1} + M_1 \boldsymbol{v}$ for some matrix T and vector $\boldsymbol{v}$. We can then compute a key for W_2:

$$\begin{aligned}
\widehat{\mathrm{SK}}_{W_2} &= (\boldsymbol{k_\rho}, k_r, \underbrace{k_{\boldsymbol{u_1}} \cdot \psi(\boldsymbol{k}_{\mathrm{del}}, v)}_{k_{\boldsymbol{u_2}}}, \underbrace{\psi^*(\boldsymbol{k}_{\mathrm{del}}, T)}_{\boldsymbol{k'}_{\mathrm{del}}}) \\
&= \left(g_1^{\boldsymbol{\rho}}, g_1^r, g_1^{\tau + r(\varphi + \langle \boldsymbol{u_1}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\rho}, \boldsymbol{\sigma} \rangle} \cdot g_1^{r\boldsymbol{v}^\top {M_1}^\top \alpha}, g_1^{r T^\top {M_1}^\top \boldsymbol{\alpha}}\right) \\
&\quad \times \left(g_3^{\boldsymbol{\mu_\rho}}, g_3^{\mu_r}, g_3^{\mu_{\boldsymbol{u_1}} + \langle \boldsymbol{v}, \boldsymbol{\mu}_{\mathrm{del}} \rangle}, g_3^{T^\top \boldsymbol{\mu}_{\mathrm{del}}}\right) \\
&= \left(g_1^{\boldsymbol{\rho}}, g_1^r, g_1^{\tau + r(\varphi + \langle \boldsymbol{u_1}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\rho}, \boldsymbol{\sigma} \rangle + r\langle M_1 \boldsymbol{v}, \boldsymbol{\alpha} \rangle}, g_1^{r {M_2}^\top \boldsymbol{\alpha}}\right) \times g_3^{\boldsymbol{\mu'}} \\
&= \left(g_1^{\boldsymbol{\rho}}, g_1^r, g_1^{\tau + r(\varphi + \langle \boldsymbol{u_2}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\rho}, \boldsymbol{\sigma} \rangle}, g_1^{r {M_2}^\top \boldsymbol{\alpha}}\right) \times g_3^{\boldsymbol{\mu'}} \\
&\quad \text{where } \boldsymbol{\mu'} = (\boldsymbol{\mu_\rho}, \mu_r, \mu_{\boldsymbol{u_1}} + \langle \boldsymbol{v}, \boldsymbol{\mu}_{\mathrm{del}} \rangle, T^\top \boldsymbol{\mu}_{\mathrm{del}}).
\end{aligned}$$

Then we pick $\Delta r \xleftarrow{R} \mathbb{Z}_N$, $\boldsymbol{\Delta\rho} \xleftarrow{R} \mathbb{Z}_N^n$ and $\boldsymbol{\Delta\mu} \xleftarrow{R} \mathbb{Z}_N^{d+n+2}$ to re-randomize it:

$$\begin{aligned}
\mathrm{SK}_{W_2} &= \hat{\mathrm{SK}}_{W_2} \times \left(g_1^{\boldsymbol{\Delta\rho}}, g_1^{\Delta r}, g_1^{\Delta r(\varphi + \langle \boldsymbol{u_2}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\Delta\rho}, \boldsymbol{\sigma} \rangle}, g_1^{\Delta r {M_2}^\top \boldsymbol{\alpha}}\right) \times g_3^{\boldsymbol{\Delta\mu}}. \\
&= \left(g_1^{\boldsymbol{\rho} + \boldsymbol{\Delta\rho}}, g_1^{r + \Delta r}, g_1^{\tau + (r + \Delta r)(\varphi + \langle \boldsymbol{u_2}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\rho} + \boldsymbol{\Delta\rho}, \boldsymbol{\sigma} \rangle}, g_1^{(r + \Delta r) {M_2}^\top \boldsymbol{\alpha}}\right) \times g_3^{\boldsymbol{\mu'} + \boldsymbol{\Delta\mu}}.
\end{aligned}$$

Notice that $\mathsf{Delegate}(W_1, \mathrm{SK}_{W_1}, W_1)$ is a re-randomization of SK_{W_1}.

$\mathsf{KeyGen}(\mathrm{SK}_E, W)$ can be seen as $\mathsf{Delegate}(E, \mathrm{SK}_E, W)$.

$\mathsf{Encrypt}(m, \boldsymbol{u})$, where m is encoded as an element of the target group $\mathbb{G}_T$, picks a random $s \xleftarrow{R} \mathbb{Z}_N^n$ and computes the ciphertext

$$\mathrm{CT} = (\boldsymbol{c_\sigma}, c_{\boldsymbol{u}}, c_s, c_m) = \left(g_1^{s\boldsymbol{\sigma}}, g_1^{-s(\varphi+\langle \boldsymbol{u},\boldsymbol{\alpha}\rangle)}, g_1^s, m \cdot t^s\right) \in \mathbb{G}^{n+2} \times \mathbb{G}_T.$$

$\mathsf{Decrypt}(\mathrm{CT}, \boldsymbol{u}, W, \mathrm{SK}_W)$ where $\mathrm{CT} = (\mathrm{CT}^*, c_m) = (\boldsymbol{c_\sigma}, c_{\boldsymbol{u}}, c_s, c_m)$ first delegates SK_W to obtain the key $\mathrm{SK}_{\boldsymbol{u}} = (\boldsymbol{k_\rho}, k_r, k_{\boldsymbol{u}}) = \left(g_1^{\boldsymbol{\rho}}, g_1^r, g_1^{\tau+r(\varphi+\langle \boldsymbol{u},\boldsymbol{\alpha}\rangle)-\langle \boldsymbol{\rho},\boldsymbol{\sigma}\rangle}\right) \times g_3^{\boldsymbol{\mu}}$. It then recovers

$$\begin{aligned}
\frac{c_m}{\hat{e}_{n+2}(\mathrm{CT}^*, \mathrm{SK}_{\boldsymbol{u}})} &= \frac{c_m}{\hat{e}_n(\boldsymbol{c_\sigma}, \boldsymbol{k_\rho}) \cdot \hat{e}(c_{\boldsymbol{u}}, k_r) \cdot \hat{e}(c_s, k_{\boldsymbol{u}})} \\
&= \frac{m \cdot t^s}{\hat{e}(g_1, g_1)^{s\langle \boldsymbol{\rho},\boldsymbol{\sigma}\rangle - rs(\varphi+\langle \boldsymbol{u},\boldsymbol{\alpha}\rangle) + s(\tau+r(\varphi+\langle \boldsymbol{u},\boldsymbol{\alpha}\rangle)-\langle \boldsymbol{\rho},\boldsymbol{\sigma}\rangle)}} \\
&= \frac{m \cdot \hat{e}(g_1, g_1)^{s\tau}}{\hat{e}(g_1, g_1)^{s\tau}} = m.
\end{aligned}$$

3.3 Security of Our Scheme

The security of our spatial encryption scheme follows from the following theorem:

Theorem 1. *Under Assumptions 1, 2, 3 and for* $\ell_{\mathrm{SK}} = (n-1-2c)\log p_2$*, where* $c > 0$ *is any fixed positive constant, the spatial encryption scheme described in Section 3.2 is* ℓ_{SK}*-master-leakage secure.*

Since the proof of Theorem 1 uses the dual system encryption technique, first developed by Waters in [Wat09], let us first recall this proof methodology.

Dual System Encryption. In a dual system encryption scheme, both ciphertexts and secret keys can take one of two indistinguishable forms: *normal*, or *semi-functional*. A normal secret key can be used to decrypt any type of ciphertext (normal or semi-functional). Likewise, a normal ciphertext can be decrypted by any type of secret key (normal or semi-functional). As shown in Table 1, the only combination that does not work is a semi-functional key with a semi-functional ciphertext.

The idea behind the proof of security of a dual system encryption scheme is to gradually modify the secret keys and ciphertexts of the system to make them *semi-functional*. When both secret keys and ciphertexts are semi-functional, proving security is straightforward.

Table 1. Different combinations of secret keys and ciphertexts

secret key	ciphertext	decryption
normal	normal	correct
normal	semi-functional	correct
semi-functional	normal	correct
semi-functional	semi-functional	incorrect

Proof Overview. In order to prove Theorem 1, we need to prove that the advantage of the adversary in the MasterLeakSpatial game is negligible. Towards this goal, we will define a sequence of games that are closely related to each other. We will prove that the difference in probability of winning for the adversary is negligible between two consecutive games of this sequence, until we get to a game where both keys and ciphertexts are semi-functional, in which the adversary only has a negligible chance of winning the game.

A *semi-functional* key has $\mathbb{G}_{p_2}$ parts, while a normal key has not. We pick at random g_2, a generator of $\mathbb{G}_{p_2}$.

$\mathsf{KeyGenSF}(\mathrm{SK}, W)$ first calls $\mathsf{Delegate}(E, \mathrm{SK}, W)$ so as to get a normal key $\mathrm{SK}_W = (\boldsymbol{k_\rho}, k_r, k_{\boldsymbol{u}}, \boldsymbol{k}_{\mathrm{del}})$. It then picks $\boldsymbol{\gamma} \xleftarrow{R} \mathbb{Z}_N^{n+2}$, $\boldsymbol{\theta} \xleftarrow{R} \mathbb{Z}_N^d$ and computes

$$\widetilde{\mathrm{SK}}_W = \left((\boldsymbol{k_\rho}, k_r, k_{\boldsymbol{u}}) \times g_2^{\boldsymbol{\gamma}}, \boldsymbol{k}_{\mathrm{del}} \times g_2^{\boldsymbol{\theta}}\right)$$

Game MasterLeakC

procedure Initialize
$(\mathrm{PP}, \mathrm{SK}) \xleftarrow{R} \mathsf{Setup}$
$\mathcal{R} \leftarrow \emptyset$
$\mathcal{T}[0] \leftarrow (E, \mathrm{SK}, 0)$
$\mathtt{nbKeys} \leftarrow 1$
$\beta \xleftarrow{R} \{0,1\}$
Return PP

procedure Create(h, W)
If $\mathcal{T}[h].\mathtt{space} = E$ then
  $\mathrm{SK}_W \xleftarrow{R} \mathsf{KeyGen}(\mathcal{T}[h].\mathtt{key}, W)$
  $h' \leftarrow \mathtt{nbKeys}$
  $\mathcal{T}[h'] \leftarrow (W, \mathrm{SK}_W, 0)$
  $\mathtt{nbKeys} \leftarrow \mathtt{nbKeys} + 1$
  Return h'
Else
  Return $\perp$

procedure Delegate(h, W')
Return **Create**$(0, W')$

procedure Leak(h, f)
$\mathrm{SK} \leftarrow \mathcal{T}[h].\mathtt{key}$
If $\mathcal{T}[h].\mathtt{leaked} + |f(\mathrm{SK})| \leq \ell_{\mathrm{SK}}$ then
  $\mathcal{T}[h].\mathtt{leaked} \leftarrow \mathcal{T}[h].\mathtt{leaked} + |f(\mathrm{SK})|$
  Return $f(\mathrm{SK})$
Else
  Return $\perp$

procedure Reveal(h)
If $\mathcal{T}[h].\mathtt{space} \neq E$ then
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{T}[h].\mathtt{space}$
  Return $\mathcal{T}[h].\mathtt{key}$
Else
  Return $\perp$

procedure LR$(\boldsymbol{u}^*, m_0, m_1)$
$\mathrm{CT}^* \xleftarrow{R}$ EncryptSF$(\boldsymbol{u}^*, m_\beta)$
Return CT^*

procedure Finalize(β')
Return $(\beta' = \beta)$

Fig. 2. Algorithms of the game MasterLeakC

Game MasterLeakCK

procedure Initialize
$(\mathrm{PP}, \mathrm{SK}) \xleftarrow{R} \mathsf{Setup}$
$\boxed{\widetilde{\mathrm{SK}} \leftarrow \mathsf{KeyGenSF}(\mathrm{SK}, E)}$
$\mathcal{R} \leftarrow \emptyset$
$\mathcal{T}[0] \leftarrow (E, \mathrm{SK}, \boxed{\widetilde{\mathrm{SK}}}, 0)$
$\mathtt{nbKeys} \leftarrow 1$
$\beta \xleftarrow{R} \{0,1\}$
Return PP

procedure Create(h, W)
If $\mathcal{T}[h].\mathtt{space} = E$ then
  $\mathrm{SK}_W \xleftarrow{R} \mathsf{KeyGen}(\mathcal{T}[h].\mathtt{key}, W)$
  $\boxed{\widetilde{\mathrm{SK}}_W \xleftarrow{R} \mathsf{KeyGenSF}(\mathcal{T}[h].\mathtt{key}, W)}$
  $h' \leftarrow \mathtt{nbKeys}$
  $\mathcal{T}[h'] \leftarrow (W, \mathrm{SK}_W, \boxed{\widetilde{\mathrm{SK}}_W}, 0)$
  $\mathtt{nbKeys} \leftarrow \mathtt{nbKeys} + 1$
  Return h'
Else
  Return $\perp$

procedure Delegate(h, W')
Return **Create**$(0, W')$

procedure Leak(h, f)
$\widetilde{\mathrm{SK}} \leftarrow \boxed{\mathcal{T}[h].\mathtt{keySF}}$
If $\mathcal{T}[h].\mathtt{leaked} + |f(\widetilde{\mathrm{SK}})| \leq \ell_{\mathrm{SK}}$ then
  $\mathcal{T}[h].\mathtt{leaked} \leftarrow \mathcal{T}[h].\mathtt{leaked} + |f(\widetilde{\mathrm{SK}})|$
  Return $f(\widetilde{\mathrm{SK}})$
Else
  Return $\perp$

procedure Reveal(h)
If $\mathcal{T}[h].\mathtt{space} \neq E$ then
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{T}[h].\mathtt{space}$
  Return $\boxed{\mathcal{T}[h].\mathtt{keySF}}$
Else
  Return $\perp$

procedure LR$(\boldsymbol{u}^*, m_0, m_1)$
$\mathrm{CT}^* \xleftarrow{R} \mathsf{EncryptSF}(\boldsymbol{u}^*, m_\beta)$
Return CT^*

procedure Finalize(β')
Return $(\beta' = \beta)$

Fig. 3. Algorithms of the game MasterLeakCK

$\mathsf{EncryptSF}(\boldsymbol{u}, m)$ first calls $\mathsf{Encrypt}(\boldsymbol{u}, m)$ to get $\mathrm{CT} = (\boldsymbol{c_\sigma}, c_{\boldsymbol{u}}, c_s, c_m)$. It then picks $\boldsymbol{\delta} \xleftarrow{R} \mathbb{Z}_N^{n+2}$ and computes the ciphertext

$$\widetilde{\mathrm{CT}} = \left((\boldsymbol{c_\sigma}, c_{\boldsymbol{u}}, c_s) \times g_2^{\boldsymbol{\delta}}, c_m\right).$$

$\boldsymbol{\gamma}, \boldsymbol{\theta}, \boldsymbol{\delta}$ are called the *semi-functional parameters* of the secret key, the delegation key and the ciphertext, respectively. If someone uses a semi-functional key $\widetilde{\mathrm{SK}}$ for space $W = \mathrm{Aff}(M, \boldsymbol{u_1})$ with parameters $(\boldsymbol{\gamma}, \boldsymbol{\theta})$ to construct a secret key for vector $\boldsymbol{u_2} = \boldsymbol{u_1} + M\boldsymbol{v}$ with KeyGen, then this will be semi-functional with parameter $\boldsymbol{\gamma'} = \boldsymbol{\gamma} + (0, \ldots, 0, \langle \boldsymbol{v}, \boldsymbol{\theta} \rangle)$.

A semi-functional secret key for vector $\boldsymbol{u}$ is called *nominal* with respect to a ciphertext for vector $\boldsymbol{u'}$ if it can correctly decrypt it, thus if and only if:

$$\langle \boldsymbol{\gamma}, \boldsymbol{\delta} \rangle = 0 \bmod p_2 \text{ and } \boldsymbol{u} = \boldsymbol{u'}$$

because we get an extra term $\hat{e}(g_2, g_2)^{\langle \boldsymbol{\gamma}, \boldsymbol{\delta} \rangle}$ by the pairing. If the secret key cannot decrypt it, it is called *truly* semi-functional.

First, we define MasterLeakSpatial*, which is the same game as MasterLeakSpatial except that all **Delegate**(h, W') calls have been replaced by **Create**$(0, W')$

	W_0	SK_{W_0} $\widetilde{\mathrm{SK}}_{W_0}$ 4	W_1	SK_{W_1} $\widetilde{\mathrm{SK}}_{W_1}$ 6	W_2	SK_{W_2} $\widetilde{\mathrm{SK}}_{W_2}$ 2	W_3	SK_{W_3} $\widetilde{\mathrm{SK}}_{W_3}$ 3
	$j-2$		$j-1$		j		$j+1$	

Fig. 4. Example of $\mathcal{T}$ array being filled in the $\mathsf{MasterLeakCK}_j$ game. Each cell contains the quadruplet (`space`, `key`, `keySF`, `leaked`). The adversary has access to the semi-functional versions of the $j-1$ first keys, and to the normal versions of the other keys.

calls. As the keys are identically distributed, the advantage of the adversary in MasterLeakSpatial is negligibly close to its advantage in MasterLeakSpatial*.

Then we define MasterLeakC in Figure 2, which is similar to MasterLeakSpatial* except that the challenge ciphertext is now semi-functional. Using Assumption 1, we will show that the advantages of the adversary in those games are negligibly close.

In MasterLeakCK, which is described in Figure 3, the semi-functional versions of the secret keys are stored in $\mathcal{T}$ as well: it contains quadruplets of the form (`space`, `key`, `keySF`, `leaked`). The **Create** calls use the normal versions of the keys while the **Leak** and **Reveal** queries apply to the semi-functional versions.

Game $\mathsf{MasterLeakCK}_j$

procedure Initialize
$(\mathrm{PP}, \mathrm{SK}) \xleftarrow{R} \mathsf{Setup}$
$\widetilde{\mathrm{SK}} \leftarrow \mathsf{KeyGenSF}(\mathrm{SK}, E)$
$\mathcal{R} \leftarrow \emptyset$
$\mathcal{T}[0] \leftarrow (E, \mathrm{SK}, \widetilde{\mathrm{SK}}, 0)$
$\mathtt{nbKeys} \leftarrow 1$
$\beta \xleftarrow{R} \{0,1\}$
Return PP

procedure Create(h, W)
If $\mathcal{T}[h].\mathtt{space} = E$ then
  $\mathrm{SK}_W \xleftarrow{R} \mathsf{KeyGen}(\mathcal{T}[h].\mathtt{key}, W)$
  $\widetilde{\mathrm{SK}}_W \xleftarrow{R} \mathsf{KeyGenSF}(\mathcal{T}[h].\mathtt{key}, W)$
  $h' \leftarrow \mathtt{nbKeys}$
  $\mathcal{T}[h'] \leftarrow (W, \mathrm{SK}_W, \widetilde{\mathrm{SK}}_W, 0)$
  $\mathtt{nbKeys} \leftarrow \mathtt{nbKeys} + 1$
  Return h'
Else
  Return $\perp$

procedure Delegate(h, W')
Return **Create**$(0, W')$

procedure Leak(h, f)
$\mathrm{SK} \leftarrow \boxed{h > j}\ ?\ \mathcal{T}[h].\mathtt{key} : \mathcal{T}[h].\mathtt{keySF}$
If $\mathcal{T}[h].\mathtt{leaked} + |f(\mathrm{SK})| \leq \ell_{\mathrm{SK}}$ then
  $\mathcal{T}[h].\mathtt{leaked} \leftarrow \mathcal{T}[h].\mathtt{leaked} + |f(\mathrm{SK})|$
  Return $f(\mathrm{SK})$
Else
  Return $\perp$

procedure Reveal(h)
If $\mathcal{T}[h].\mathtt{space} \neq E$ then
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{T}[h].\mathtt{space}$
  Return $\boxed{h > j}\ ?\ \mathcal{T}[h].\mathtt{key} : \mathcal{T}[h].\mathtt{keySF}$
Else
  Return $\perp$

procedure LR$(\boldsymbol{u}^*, m_0, m_1)$
$\mathrm{CT}^* \xleftarrow{R} \mathsf{EncryptSF}(\mathrm{PP}, \boldsymbol{u}^*, m_\beta)$
Return CT^*

procedure Finalize(β')
Return $(\beta' = \beta)$

Fig. 5. Algorithms of the game $\mathsf{MasterLeakCK}_j$

We will need Assumption 3 to prove that the advantage of the adversary is negligible in this game.

In the $\mathsf{MasterLeakCK}_j$ game, which is described in Figure 5, for the $j-1$ first keys the attacker creates, it will have access to the semi-functional versions of them via **Leak** or **Reveal** queries. For the other keys, it will have access to the normal versions (see also Figure 4). Notice that $\mathsf{MasterLeakCK}_0$ is exactly MasterLeakC while $\mathsf{MasterLeakCK}_{\mathtt{nbKeys}}$ is exactly MasterLeakCK. We will show that if Assumption 2 holds, the difference of probability of winning for the adversary in $\mathsf{MasterLeakCK}_j$ versus $\mathsf{MasterLeakCK}_{j+1}$ is negligible. To conclude the proof, we show that the adversary's probability of winning in MasterLeakCK is negligible as long as Assumption 3 holds.

4 Concluding Remarks

We have designed the first leakage-resilient spatial encryption scheme, which is an important instance of generalized identity-based encryption. Our scheme is proven secure in the bounded-leakage model. To obtain stronger security guarantees, we need to consider models such as continual leakage and tolerate leakage during key updates. To address this problem, a promising direction would be to adapt the result of [LLW11] to the spatial encryption scenario.

Acknowledgements. This work was supported in part by the French ANR-10-SEGI-015 PRINCE Project and by the European Commission through the ICT Program under Contract ICT-2007-216676 ECRYPT II.

References

[AGV09] Akavia, A., Goldwasser, S., Vaikuntanathan, V.: Simultaneous Hardcore Bits and Cryptography against Memory Attacks. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 474–495. Springer, Heidelberg (2009)

[BFO08] Boldyreva, A., Fehr, S., O'Neill, A.: On Notions of Security for Deterministic Encryption, and Efficient Constructions without Random Oracles. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 335–359. Springer, Heidelberg (2008)

[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)

[BGW05] Boneh, D., Gentry, C., Waters, B.: Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 258–275. Springer, Heidelberg (2005)

[BH08] Boneh, D., Hamburg, M.: Generalized Identity Based and Broadcast Encryption Schemes. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 455–470. Springer, Heidelberg (2008)

[BKKV10] Brakerski, Z., Kalai, Y.T., Katz, J., Vaikuntanathan, V.: Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In: FOCS Annual Symposium on Foundations of Computer Science, pp. 501–510. IEEE Computer Society (2010)

[Bon07] Boneh, D.: Bilinear Groups of Composite Order (Invited Talk). In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, p. 1. Springer, Heidelberg (2007)

[BSW11] Boyle, E., Segev, G., Wichs, D.: Fully Leakage-Resilient Signatures. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 89–108. Springer, Heidelberg (2011)

[DHLAW10] Dodis, Y., Haralambiev, K., López-Alt, A., Wichs, D.: Cryptography against continuous memory attacks. In: FOCS Annual Symposium on Foundations of Computer Science, pp. 511–520. IEEE Computer Society Press (2010)

[HL02] Horwitz, J., Lynn, B.: Toward Hierarchical Identity-Based Encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 466–481. Springer, Heidelberg (2002)

[LLW11] Lewko, A.B., Lewko, M., Waters, B.: How to leak on key updates. In: 43rd ACM STOC Annual ACM Symposium on Theory of Computing, pp. 725–734. ACM Press (2011)

[LOS+10] Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010)

[LRW11] Lewko, A., Rouselakis, Y., Waters, B.: Achieving Leakage Resilience through Dual System Encryption. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 70–88. Springer, Heidelberg (2011)

[LW10] Lewko, A., Waters, B.: New Techniques for Dual System Encryption and Fully Secure HIBE with Short Ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010)

[MD11] Moriyama, D., Doi, H.: A fully secure spatial encryption scheme. IEICE Transactions 94-A(1), 28–35 (2011)

[Wat09] Waters, B.: Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009)

A Security Proof of Theorem 1

A.1 Lemma for Leakage Analysis

Our analysis of the leakage resilience of our system will rely on the following lemma from [BKKV10, LRW11], which is proven in [BFO08]. Below, we let $dist(X_1, X_2)$ denote the statistical distance of two random variables X_1 and X_2.

Lemma 2. *Let $m, l, d \in \mathbb{N}$, $m \geq l \geq 2d$ and let p be a prime. Let $X \xleftarrow{R} \mathbb{Z}_p^{m \times \ell}$, let $Y \xleftarrow{R} \mathbb{Z}_p^{m \times d}$, and let $T \xleftarrow{R} Rk_d(\mathbb{Z}_p^{\ell \times d})$, where $Rk_d(\mathbb{Z}_p^{\ell \times d})$ denotes the set of $\ell \times d$ matrices of tank d with entries in $\mathbb{Z}_p$. Let $f : \mathbb{Z}_p^{m \times d} \to W$ be some function. Then,*

$$dist((X, f(X \cdot T)), (X, f(Y))) \leq \varepsilon$$

as long as

$$|W| \leq 4 \cdot \left(1 - \frac{1}{p}\right) \cdot p^{\ell-(2d-1)} \cdot \varepsilon^2.$$

More precisely, we will use the following corollary.

Corollary 3. Let $m \in \mathbb{N}, m \geq 3$, and let p be a prime. Let $\boldsymbol{\Delta} \xleftarrow{R} \mathbb{Z}_p^m$, $\boldsymbol{\nu} \xleftarrow{R} \mathbb{Z}_p^m$ and let $\boldsymbol{\nu'}$ be chosen uniformly randomly from the set of vectors in $\mathbb{Z}_p^m$ which are orthogonal to $\boldsymbol{\Delta}$ under the dot product modulo p. Let $f : \mathbb{Z}_p^m \to W$ be some function. Then,

$$dist\left((\boldsymbol{\Delta}, f(\boldsymbol{\nu})), (\boldsymbol{\Delta}, f(\boldsymbol{\nu'}))\right) \leq \varepsilon$$

as long as

$$|W| \leq 4 \cdot \left(1 - \frac{1}{p}\right) \cdot p^{m-2} \cdot \varepsilon^2.$$

Proof. We apply Lemma 2 with $d = 1$ and $\ell = m - 1$. Y then corresponds to $\boldsymbol{\nu}$, while X corresponds to a basis of the orthogonal space of $\boldsymbol{\Delta}$. We note that $\boldsymbol{\nu'}$ is then distributed as $X \cdot T$, where $T \xleftarrow{R} Rk_1(\mathbb{Z}_p^{m-1\times 1})$. We note that X is determined by $\boldsymbol{\Delta}$, and is distributed as $X \xleftarrow{R} \mathbb{Z}_p^{m\times m-1}$, since $\boldsymbol{\Delta}$ is chosen uniformly randomly from $\mathbb{Z}_p^m$. It follows that

$$dist\left((\boldsymbol{\Delta}, f(\boldsymbol{\nu})), (\boldsymbol{\Delta}, f(\boldsymbol{\nu'}))\right) = dist((X, f(X \cdot T)), (X, f(Y))) \leq \varepsilon.$$

This corollary allows us to set $\ell_{\mathrm{SK}} = (n - 1 - 2c) \log p_2$ for our construction (we will have $n + 1 = m$), where c is any fixed positive constant (so that $\varepsilon = p_2^{-c}$ is negligible).

A.2 Proof

The proof of Theorem 1 follows easily from Lemmas 5, 6, and 9, and is largely based on that of [LRW11].

Table 2. Assumptions that will be used for the proofs.

game	game	proof
MasterLeakSpatial	MasterLeakSpatial*	Lemma 4
MasterLeakSpatial*	MasterLeakC	Lemma 5
MasterLeakCK$_j$	MasterLeakCK$_{j+1}$	Lemma 6
MasterLeakCK		Lemma 9

Lemma 4. *Any polynomial-time attacker $\mathcal{A}$ has an identical probability of winning in* MasterLeakSpatial *versus* MasterLeakSpatial*.

Proof. It is easy to verify that the output of $\mathsf{Delegate}(W_1, \mathrm{SK}_{W_1}, W_2)$ is identically distributed to the output of $\mathsf{KeyGen}(\mathrm{SK}_E, W_2)$.

Lemma 5. *If Assumption 1 holds, any polynomial-time attacker $\mathcal{A}$ has only a negligibly different probability of winning in* MasterLeakSpatial* *versus* MasterLeakC.

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which attains a non-negligible difference in probability of winning between those two games. We will build a PPT algorithm $\mathcal{B}$ that breaks Assumption 1 with non-negligible advantage.

$\mathcal{B}$ receives $\mathcal{D}_1 = (N, \mathbb{G}, \mathbb{G}_T, \hat{e}, g_1, g_3)$ and a challenge term $T \in \mathbb{G}_{p_1 p_2^2}$. Then it plays the MasterLeakSpatial* or the MasterLeakC game with $\mathcal{A}$ in the following way.

Setup. $\mathcal{B}$ picks $(\tau, \boldsymbol{\sigma}, \boldsymbol{\alpha}, \varphi) \xleftarrow{R} \mathbb{Z}_N \times \mathbb{Z}_N^n \times \mathbb{Z}_N^n \times \mathbb{Z}_N$ then gives

$$\mathrm{PP} = (N, g_1, g_3, g_1^{\varphi}, g_1^{\boldsymbol{\alpha}}, \hat{e}(g_1, g_1)^{\tau}, g_1^{\boldsymbol{\sigma}})$$

to $\mathcal{A}$, where N, g_1 and g_3 are given by the challenger.

Phase 1. Knowing $\boldsymbol{\alpha}$, the simulator can generate a secret key for the whole space E and use it to execute all secret queries (**Create**, **Leak**, **KeyGen**).

Challenge. The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages m_0 and m_1 and a challenge vector $\boldsymbol{u}^*$. The simulator $\mathcal{B}$ chooses $\beta \xleftarrow{R} \{0, 1\}$ and outputs the ciphertext

$$\mathrm{CT} = (\boldsymbol{c}_{\boldsymbol{\sigma}}, c_{\boldsymbol{u}^*}, c_s, c_m) = (T^{\boldsymbol{\sigma}}, T^{-(\varphi + \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle)}, T, m_\beta \cdot \hat{e}(T, g_1^{\tau})).$$

Phase 2. $\mathcal{B}$ works the same way as in **Phase 1**.

If $T = g_1^a g_2^b$, then the ciphertext is semi-functional. This implicitly sets

$$s = a \quad \text{and} \quad \boldsymbol{\delta} = b(\boldsymbol{\sigma}, -\varphi - \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle, 1).$$

s is properly distributed since $a \xleftarrow{R} \mathbb{Z}_N$ according to the assumption. $\boldsymbol{\delta}$ is properly distributed in the attacker's view because the factors $\sigma, -\varphi - \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle$ are only seen modulo p_1 in the public parameters and not modulo p_2. Thus they are random modulo p_2 in $\mathcal{A}$'s view. Therefore, $\mathcal{B}$ has properly simulated the MasterLeakC game.

If $T = g_1^a$, the ciphertext is normal since it has no $\mathbb{G}_{p_2}$ parts. In this case, $\mathcal{B}$ has properly simulated the MasterLeakSpatial* game.

Guess. If $\mathcal{A}$ wins ($\beta' = \beta$), $\mathcal{B}$ guesses $T \in \mathbb{G}_{p_1}$. If $\mathcal{A}$ loses, $\mathcal{B}$ guesses $T \in \mathbb{G}_{p_1 p_2}$. As $\mathcal{A}$ has a non-negligible difference in probability of winning between those two games, $\mathcal{B}$ will have a non-negligible advantage in breaking Assumption 1.

Lemma 6. *If Assumption 2 holds, any polynomial-time attacker $\mathcal{A}$ has only a negligibly different probability of winning in* MasterLeakCK$_j$ *versus* MasterLeakCK$_{j+1}$.

Proof. $\mathcal{B}$ receives $\mathcal{D}_2 = (N, \mathbb{G}, \mathbb{G}_T, \hat{e}, g_1, g_3, g_1^x g_2^y, g_2^{y'} g_3^{z'})$ and a challenge term $T \in \mathbb{G}_{p_1 p_2^? p_3}$. Then it plays the $\mathsf{MasterLeakCK}_j$ or the $\mathsf{MasterLeakCK}_{j+1}$ game with $\mathcal{A}$ as follows.

Setup. $\mathcal{B}$ picks $(\tau, \boldsymbol{\sigma}, \boldsymbol{\alpha}, \varphi) \xleftarrow{R} \mathbb{Z}_N \times \mathbb{Z}_N^n \times \mathbb{Z}_N^n \times \mathbb{Z}_N$ then gives

$$\mathrm{PP} = (N, g_1, g_3, g_1^{\varphi}, g_1^{\boldsymbol{\alpha}}, \hat{e}(g_1, g_1)^{\tau}, g_1^{\boldsymbol{\sigma}})$$

to $\mathcal{A}$, where N, g_1 and g_3 are given by the challenger.

Phase 1.

- For the first $j-1$ keys, $\mathcal{B}$ picks $(\boldsymbol{\rho'}, r, \boldsymbol{\mu}) \xleftarrow{R} \mathbb{Z}_N^n \times \mathbb{Z}_N \times \mathbb{Z}_N^{d+n+2}$ and, for the semi-functional parameters, $(\boldsymbol{\gamma'}, \boldsymbol{\theta'}) \xleftarrow{R} \mathbb{Z}_N^{n+2} \times \mathbb{Z}_N^d$. It then computes:

  $$\mathrm{SK} = \left(\left(g_1^{\boldsymbol{\rho'}}, g_1^r, g_1^{\tau + r(\varphi + \langle \boldsymbol{u}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\rho'}, \boldsymbol{\sigma} \rangle} \right) \times (g_2^{y'} g_3^{z'})^{\boldsymbol{\gamma'}}, g_1^{r M^{\top} \boldsymbol{\alpha}} \times (g_2^{y'} g_3^{z'})^{\boldsymbol{\theta'}} \right) \times g_3^{\boldsymbol{\mu}}$$

  The $\mathbb{G}_{p_1}$ and $\mathbb{G}_{p_3}$ parts are properly distributed and the semi-functional parameters are $\boldsymbol{\gamma} = y' \boldsymbol{\gamma'}$ and $\boldsymbol{\theta} = y' \boldsymbol{\theta'}$. These are properly distributed as well.
- For the j-th key, $\mathcal{B}$ will use the challenge term to generate the secret key. It picks $(\boldsymbol{\rho'}, \boldsymbol{\mu}) \xleftarrow{R} \mathbb{Z}_N^n \times \mathbb{Z}_N^{d+n+2}$ and outputs

  $$\mathrm{SK} = \left(T^{\boldsymbol{\rho'}}, T, g_1^{\tau} T^{\varphi + \langle \boldsymbol{u}, \boldsymbol{\alpha} \rangle - \langle \boldsymbol{\rho'}, \boldsymbol{\sigma} \rangle}, T^{M^{\top} \boldsymbol{\alpha}} \right) \times g_3^{\boldsymbol{\mu}}$$

  If $T = g_1^a g_2^b g_3^c$, then this key is semi-functional with

  $$r = a \qquad \boldsymbol{\rho} = a \boldsymbol{\rho'} \qquad \boldsymbol{\gamma} = b\,(\boldsymbol{\rho'}, 1, \varphi + \langle \boldsymbol{u}, \boldsymbol{\alpha} \rangle - \langle \boldsymbol{\rho'}, \boldsymbol{\sigma} \rangle) \qquad \boldsymbol{\theta} = b M^{\top} \boldsymbol{\alpha}.$$

  Since the public parameters only determine φ and $\boldsymbol{\alpha}$ modulo p_1, the semi-functional parameters are random modulo p_2 in $\mathcal{A}$'s view.
- From the $j+1$ key, $\mathcal{B}$ picks $(\boldsymbol{\rho'}, r, \boldsymbol{\mu}) \xleftarrow{R} \mathbb{Z}_N^n \times \mathbb{Z}_N \times \mathbb{Z}_N^{d+n+2}$ and computes:

  $$\mathrm{SK} = \left(g_1^{\boldsymbol{\rho'}}, g_1^r, g_1^{\tau + r(\varphi + \langle \boldsymbol{u}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\rho'}, \boldsymbol{\sigma} \rangle}, g_1^{r M^{\top} \boldsymbol{\alpha}} \right) \times g_3^{\boldsymbol{\mu}}$$

Challenge. In this phase, $\mathcal{B}$ has to create a semi-functional ciphertext with $\mathsf{EncryptSF}$. $\mathcal{A}$ gives $\mathcal{B}$ two messages m_0 and m_1 and a challenge vector $\boldsymbol{u}^*$. $\mathcal{B}$ chooses $\beta \xleftarrow{R} \{0, 1\}$ and outputs the ciphertext

$$\begin{aligned} \widetilde{\mathrm{CT}} &= (\boldsymbol{c}_{\boldsymbol{\sigma}}, c_{\boldsymbol{u}^*}, c_s, c_m) \\ &= \left((g_1^x g_2^y)^{\boldsymbol{\sigma}}, (g_1^x g_2^y)^{-(\varphi + \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle)}, (g_1^x g_2^y), m_\beta \cdot \hat{e}((g_1^x g_2^y), g_1^{\tau}) \right). \end{aligned}$$

Phase 2. $\mathcal{B}$ works the same way as in **Phase 1**.

Guess. If $\mathcal{A}$ wins, $\mathcal{B}$ guesses $T \in \mathbb{G}_{p_1 p_3}$. If $\mathcal{A}$ loses, $\mathcal{B}$ guesses $T \in \mathbb{G}_{p_1 p_2 p_3}$.

The ciphertext parameters are:

$$s = x \quad \text{and} \quad \boldsymbol{\delta} = y(\boldsymbol{\sigma}, -\varphi - \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle, 1).$$

s is properly distributed, but the semi-functional parameters are not: if the space of the j-th key contains the challenge vector $\boldsymbol{u}^*$, the secret key is nominal with respect to the ciphertext. Indeed:

$$\begin{aligned}
&(\boldsymbol{\gamma} + (0, \ldots, 0, \langle \boldsymbol{v}, \boldsymbol{\theta} \rangle)) \cdot \boldsymbol{\delta} \\
&= b \left(\boldsymbol{\rho'}, 1, \varphi + \langle \boldsymbol{u}, \boldsymbol{\alpha} \rangle - \langle \boldsymbol{\rho'}, \boldsymbol{\sigma} \rangle + \left\langle \boldsymbol{v}, M^\top \boldsymbol{\alpha} \right\rangle \right) \cdot y(\boldsymbol{\sigma}, -\varphi - \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle, 1) \\
&= by \left(\langle \boldsymbol{\rho'}, \boldsymbol{\sigma} \rangle - \varphi - \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle + \varphi + \langle \boldsymbol{u}, \boldsymbol{\alpha} \rangle - \langle \boldsymbol{\rho'}, \boldsymbol{\sigma} \rangle + \langle M\boldsymbol{v}, \boldsymbol{\alpha} \rangle \right) \\
&= by \langle \boldsymbol{u} + M\boldsymbol{v} - \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle \\
&= 0 \pmod{p_2}
\end{aligned}$$

Notice that if the space of the j-th key contains a vector $\boldsymbol{w}$ which is equal to $\boldsymbol{u}^*$ modulo p_2, we obtain the same result. With the help of two lemmas, we will show that the change in the adversary's advantage is negligible.

Lemma 7. *If Assumption 2 holds, then for any PPT adversary $\mathcal{A}$, $\mathcal{A}$'s advantage in the $\mathsf{MasterLeakCK}_j$ game, or in the $\mathsf{MasterLeakCK}_{j+1}$ game, changes only by a negligible amount if we restrict it to make queries only on the challenge vector and on spaces that do not contain a vector equal to the challenge vector $\boldsymbol{u}^*$ modulo p_2.*

Proof. If there exists an adversary $\mathcal{A}$ whose advantage changes by a non-negligible amount under this restriction, then with non-negligible probability, $\mathcal{A}$ issues a query on $W = \mathrm{Aff}(M, \boldsymbol{u})$ which contains $\boldsymbol{u'}$ that satisfies $\boldsymbol{u'} \equiv \boldsymbol{u}^* \pmod{p_2}$.

Let us consider the equation $\boldsymbol{u} + M\boldsymbol{x} = \boldsymbol{u}^*$ for the unknown $\boldsymbol{x}$. It has no solution modulo N (because $\boldsymbol{u}^* \notin W$) but it has a solution modulo p_2 (because $\boldsymbol{u}^* \equiv \boldsymbol{u'} \in W \pmod{p_2}$).

By Gaussian elimination, we can reduce the equation $M\boldsymbol{x} = \boldsymbol{y}$ (where $\boldsymbol{y} = \boldsymbol{u}^* - \boldsymbol{u}$) to the following form:

$$\begin{pmatrix} 1 & * & * \\ & \ddots & * \\ & & 1 \\ & & \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ \\ \vdots \\ y_n \end{pmatrix}$$

The first d row vectors are not equal to 0 because the d column vectors of M are independent. The $n - d$ remaining rows are filled with 0's ($d < n$). As this equation has no solution modulo N, there exists some $i \in [|d+1, n|]$ such that $y_i \not\equiv 0 \pmod{N}$. But as this equation has a solution modulo p_2, we have $y_i \equiv 0 \pmod{p_2}$.

Thus we can find a non-trivial factor of N with non-negligible probability: we denote $p = \gcd(y_i, N)$ and $p' = \frac{n}{p}$. If $p' = p_1$ or $p' = p_1 p_3$, we can break

Assumption 1 by computing $T^{p'}$: if it equals 1, then $T \in \mathbb{G}_{p_1}$, else $T \in \mathbb{G}_{p_2}$. Otherwise, $p' = p_3$ and we can break Assumption 3 by computing $\hat{e}(T^{p'}, g_2^{y'} g_3^{z'})$: if it equals 1, then $T \in \mathbb{G}_{p_1 p_3}$ ($T^{p'}$ has no $\mathbb{G}_{p_2}$ parts), else $T \in \mathbb{G}_{p_1 p_2 p_3}$.

Lemma 8. *We suppose the leakage is at most $\ell_{\mathrm{SK}} = (n - 1 - 2c) \log p_2$, where $c > 0$ is a positive constant. Then, for any PPT adversary $\mathcal{A}$, $\mathcal{A}$'s advantage in the $\mathsf{MasterLeakCK}_{j+1}$ game changes only by a negligible amount when the truly semi-functional j-th key is replaced by a nominal semi-functional key whenever $\mathcal{A}$ declares the j-th key to be associated to a space that contains the challenge ciphertext vector.*

Proof. Let us suppose there exists a PPT algorithm $\mathcal{A}$ whose advantage changes by a non-negligible amount when the $\mathsf{MasterLeakCK}_{j+1}$ game changes as described above. Using $\mathcal{A}$, we will create a PPT algorithm $\mathcal{B}$ which will distinguish between $(\boldsymbol{\Delta}, f(\boldsymbol{\nu}))$ and $(\boldsymbol{\Delta}, f(\boldsymbol{\nu'}))$ from Corollary 3 with non-negligible advantage (when $m = n + 1$ and $p = p_2$). This will yield a contradiction, since these distributions have a negligible statistical distance.

$\mathcal{B}$ simulates $\mathsf{MasterLeakCK}_{j+1}$ as follows. It runs **Setup** and gives $\mathcal{A}$ the public parameters. Since $\mathcal{B}$ knows $\boldsymbol{\alpha}$ and generators of all the subgroups, it can respond to $\mathcal{A}$'s queries in **Phase 1**.

With non-negligible probability, the j-th key $\mathcal{A}$ chooses in **Phase 1** must be associated to a space that contains $\boldsymbol{u}^*$. (If it only did this with negligible probability, then the difference in advantages whenever it happens would be negligible.)

$\mathcal{B}$ will not create that key but instead will encode the leakage $\mathcal{A}$ asks for, as a single polynomial-time computable function f with domain $\mathbb{Z}_{p_2}^{n+1}$ and with an image of size $2^{\ell_{\mathrm{SK}}}$. $\mathcal{B}$ receives a sample $(\boldsymbol{\Delta}, f(\boldsymbol{\Gamma}))$, where $\boldsymbol{\Gamma} = (\Gamma_1, \ldots, \Gamma_{n+1})$ is either distributed as $\boldsymbol{\nu}$ or as $\boldsymbol{\nu'}$. $\mathcal{B}$ will use $f(\Gamma)$ to answer all of $\mathcal{A}$'s leakage queries on the j-th key by implicitly defining this key as follows.

$\mathcal{B}$ chooses $(r_1, r_2, \boldsymbol{\theta}) \xleftarrow{R} \mathbb{Z}_{p_2}^{d+2}$. We let g_2 denote a generator of $\mathbb{G}_{p_2}$. $\mathcal{B}$ implicitly sets the $\mathbb{G}_{p_2}$ components of the key to be $g_2^{\boldsymbol{\Gamma'}}$, where $\boldsymbol{\Gamma'}$ is defined to be:

$$\boldsymbol{\Gamma'} = (\Gamma_1, \ldots, \Gamma_n, \Gamma_{n+1} + r_1, r_2, \boldsymbol{\theta}).$$

$\mathcal{B}$ defines the other components of the key to fit their appropriate distribution.

At some point, $\mathcal{A}$ declares the challenge vector $\boldsymbol{u}^*$. If the space of the j-th key does not contain $\boldsymbol{u}^*$, then $\mathcal{B}$ aborts the simulation and guesses whether $\boldsymbol{\Gamma}$ is orthogonal to $\boldsymbol{\Delta}$ randomly. However, the simulation continues with non-negligible probability. Suppose the space of the j-th key is $W = \mathrm{Aff}(M, \boldsymbol{u})$ and let $\boldsymbol{v}$ be the vector such that $\boldsymbol{u}^* = \boldsymbol{u} + M\boldsymbol{v}$.

$\mathcal{B}$ chooses a random element $t_2 \xleftarrow{R} \mathbb{Z}_{p_2}$ such that $\Delta_{n+1} r_1 + (r_2 + \langle \boldsymbol{v}, \boldsymbol{\theta} \rangle) t_2 = 0 \pmod{p_2}$. It then constructs the challenge ciphertext, using $(\boldsymbol{\Delta}, t_2) \in \mathbb{Z}_{p_2}^{n+2}$ as parameter.

If $\boldsymbol{\Gamma}$ is orthogonal to $\boldsymbol{\Delta}$, then the j-th key is nominally semi-functional (and well distributed):

$$\begin{aligned}&((\Gamma_1, \ldots, \Gamma_n, \Gamma_{n+1} + r_1, r_2) + (0, \ldots, 0, \langle \boldsymbol{v}, \boldsymbol{\theta} \rangle)) \cdot (\boldsymbol{\Delta}, t_2)\\ &= \boldsymbol{\Gamma} \cdot \boldsymbol{\Delta} + \Delta_{n+1} r_1 + (r_2 + \langle \boldsymbol{v}, \boldsymbol{\theta} \rangle) t_2\\ &= 0 \pmod{p_2}.\end{aligned}$$

If $\boldsymbol{\Gamma}$ is not orthogonal to $\boldsymbol{\Delta}$, then the challenge key is truly semi-functional (and also well distributed).

It is clear that $\mathcal{B}$ can easily handle **Phase 2** queries, since the j-th key cannot be queried when its space contains $\boldsymbol{u}^*$. Therefore, $\mathcal{B}$ can use the output of $\mathcal{A}$ to gain a non-negligible advantage in distinguishing $(\boldsymbol{\Delta}, f(\boldsymbol{\nu}))$ and $(\boldsymbol{\Delta}, f(\boldsymbol{\nu'}))$, which violates Corollary 3.

The above lemmas conclude the proof of Lemma 6.

Lemma 9. *If Assumption 3 holds, any polynomial-time attacker $\mathcal{A}$ has only a negligible advantage in* MasterLeakCK.

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ which attains a non-negligible advantage in MasterLeakCK. We will build a PPT algorithm $\mathcal{B}$ that breaks Assumption 3 with non-negligible advantage.

$\mathcal{B}$ receives $\mathcal{D}_3 = (N, \mathbb{G}, \mathbb{G}_T, \hat{e}, g_1, g_2, g_3, g_1^\tau g_2^y, g_1^{x'} g_2^{y'})$ and a challenge term T which is either $\hat{e}(g_1, g_1)^{\tau x'}$ or a random term of $\mathbb{G}_T$. Algorithm $\mathcal{B}$ works as follows.

Setup. $\mathcal{B}$ picks $(\boldsymbol{\sigma}, \boldsymbol{\alpha}, \varphi) \xleftarrow{R} \mathbb{Z}_N^n \times \mathbb{Z}_N^n \times \mathbb{Z}_N$. Notice that now τ is unknown. The term $\hat{e}(g_1, g_1)^\tau$ is computed as $\hat{e}(g_1^\tau g_2^y, g_1)$. It gives $\mathrm{PP} = (N, g_1, g_3, g_1^\varphi, g_1^{\boldsymbol{\alpha}}, \hat{e}(g_1, g_1)^\tau, g_1^{\boldsymbol{\sigma}})$ to $\mathcal{A}$, where N, g_1 and g_3 are given by the challenger.

Phase 1. For each secret key requested by the adversary, the simulator $\mathcal{B}$ picks the random exponents $(r, \boldsymbol{\rho}, \boldsymbol{\mu}) \xleftarrow{R} \mathbb{Z}_N \times \mathbb{Z}_N^n \times \mathbb{Z}_N^{d+n+2}$ and for the semi-functional parameters, $\boldsymbol{\gamma'} \xleftarrow{R} \mathbb{Z}_N^{n+2}$ and $\boldsymbol{\theta} \xleftarrow{R} \mathbb{Z}_N^d$. It uses the secret key

$$\mathrm{SK} = \left(\left(g_1^{\boldsymbol{\rho}}, g_1^r, (g_1^\tau g_2^y) g_1^{r(\varphi + \langle \boldsymbol{u}, \boldsymbol{\alpha} \rangle) - \langle \boldsymbol{\rho'}, \boldsymbol{\sigma} \rangle} \right) \times g_2^{\boldsymbol{\gamma'}}, g_1^{r M^\top \boldsymbol{\alpha}} \times g_2^{\boldsymbol{\theta}} \right) \times g_3^{\boldsymbol{\mu}}$$

It is a properly distributed semi-functional key with parameters

$$\boldsymbol{\gamma} = \boldsymbol{\gamma'} + (0, \ldots, 0, y) \text{ and } \boldsymbol{\theta}.$$

Challenge. The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages m_0, m_1 and a challenge vector $\boldsymbol{u}^*$. The simulator chooses $\beta \xleftarrow{R} \{0, 1\}$ and outputs the following ciphertext:

$$\mathrm{CT} = \left(\left(g_1^{x'} g_2^{y'} \right)^{\boldsymbol{\sigma}}, \left(g_1^{x'} g_2^{y'} \right)^{-\varphi - \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle}, \left(g_1^{x'} g_2^{y'} \right), m_\beta \cdot T \right).$$

Phase 2. $\mathcal{B}$ works the same way as in **Phase 1**.

If $T = \hat{e}(g_1, g_1)^{\tau x'}$, then we get a semi-functional ciphertext of m_β with parameters

$$s = x' \quad \text{and} \quad \boldsymbol{\delta} = y'(\boldsymbol{\sigma}, -\varphi - \langle \boldsymbol{u}^*, \boldsymbol{\alpha} \rangle, 1).$$

$\boldsymbol{\delta}$ is properly distributed since all terms are random modulo p_2. Therefore, $\mathcal{B}$ has properly simulated game MasterLeakCK.

If $T \xleftarrow{R} \mathbb{G}_T$, then c_m is entirely random and we get a semi-functional ciphertext of a random message. Hence, the value of β is information-theoretically hidden and the probability of success of $\mathcal{A}$ is exactly $1/2$, since $\beta \xleftarrow{R} \{0, 1\}$. Therefore, $\mathcal{B}$ can use the output of $\mathcal{A}$ to break Assumption 3 with non-negligible advantage.

Guess. If $\mathcal{A}$ wins, $\mathcal{B}$ guesses $T = \hat{e}(g_1, g_1)^{\tau x'}$. If $\mathcal{A}$ loses, $\mathcal{B}$ guesses $T \xleftarrow{R} \mathbb{G}_T$.As $\mathcal{A}$ has a non-negligible difference in probability of winning between those two games, $\mathcal{B}$ will have a non-negligible advantage in breaking Assumption 3.

This concludes the proof of Lemma 9.

On the Pseudorandom Function Assumption in (Secure) Distance-Bounding Protocols

PRF-ness alone Does Not Stop the Frauds!

Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay

Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
{ioana.boureanu,katerina.mitrokotsa,serge.vaudenay}@epfl.ch

Abstract. In this paper, we show that many formal and informal security results on distance-bounding (DB) protocols are incorrect/ incomplete. We identify that this inadequacy stems from the fact that the pseudorandom function (PRF) assumption alone, invoked in many security claims, is insufficient. To this end, we identify two distinct shortcomings of invoking the PRF assumption alone: one leads to distance-fraud attacks, whilst the other opens for man-in-the-middle (MiM) attacks. First, we describe –in a more unitary, formal fashion– why assuming that a family of functions classically used inside DB protocols is solely a PRF is unsatisfactory and what generic security flaws this leads to. Then, we present concrete constructions that disprove the PRF-based claimed security of several DB protocols in the literature; this is achieved by using some PRF programming techniques. Whilst our examples may be considered contrived, the overall message is clear: the PRF assumption should be strengthened in order to attain security against distance-fraud and MiM attacks in distance-bounding protocols!

1 Introduction

Distance-bounding (DB) protocols were introduced by Brands and Chaum [3] with the view of combating man-in-the-middle attacks against ATM systems. The main idea of DB protocols is that a tag (RFID card, smart card, etc.) should prove a short distance between them and a reader, and –most often than not– authenticate themselves in front of this reader. The authentication part is based on a pre-established secret. By default, this shared secret is a key hard-coded on the tag which the reader associates to the tag's id via a stored database. The tag is often referred to as the prover whereas the reader is referred to as a verifier. In the vast literature covering such protocols (e.g., [10,12,14,16]), three *main/classical* types of possible attacks have been distinguished. The first is *distance-fraud* (DF), in which a prover tries to convince that he is closer than what he really is. The second type of attack is the *mafia-fraud* (MF) attack, which involves three entities: an honest prover, an honest verifier and an adversary. The adversary communicates with both the prover and the verifier and tries to demonstrate to the verifier that the prover is in the verifier's proximity although

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 100–120, 2012.

the prover is in reality far away from the verifier. Finally, the third type of attack is denoted as *terrorist-fraud* (TF). Here, the adversary has the same goal as in the mafia-fraud attack, but in this case the prover is dishonest and colludes with the adversary up to the non-disclosure of essential information, i.e., secret keys or any other information that may more easily facilitate later impersonations of the tag. Other more generic MiM attacks have been imagined [4,6], generalising mainly distance-fraud or mafia-fraud respectively.

Meant to protect against such intricate attacks [3], implemented versions of DB protocols have only proven to be efficient in preventing relay attacks [5]. This is undeniably an important step. However, given the clear view of progressing in secure remote unlocking (e.g., [8]), distance-bounding protocols should be designed to resist against more *generic* (MiM, DF, TF,etc.) attacks, as aimed [3]. Whilst some attempts of formal models and formal proofs of security have recently arisen [1,6], provably secure distance-bounding is not at all a stable, well-founded area. For instance, we consider that [1], addressing the protection against terrorist-fraud using secret sharing schemes, only provides rather heuristic security analyses, failing to pinpoint the (necessary and) sufficient conditions for preventing TF on distance-bounding. On a parallel front, the model of Dürholz *et al.* in [6] is more attentive to detail, moving closer to provably secure DB. But, whilst [6,7] claim some security results, we believe that their informal proofs of security for DF and for MF are flawed. Thus, it is therein common to replace a PRF by a random function in a game-reduction proof, even if the PRF key is held by the adversary. This practice is obviously flawed. In this paper, we will show where these formal or informal proofs fell short of the correct arguments. Whilst we leave the concrete amendments of these issues for future work, we underline some concrete aspects that the state-of-art on (secure) DB has overlooked in their assessments, aspects that fundamentally compromise the security of these protocols. We formalise these concerns. We provide supporting examples, using PRF programming techniques, on a list of (claimed-to-be-secure) DB protocols[1] in the literature. We suspect that there are many more DB protocols susceptible to the kind of attacks we exhibit, especially since the DB protocols bear clear resemblances amongst them; The list of attacks herein is summarised in Table 1; as one can see, it comprises the somewhat popular DB protocols.

Table 1. Protocols Broken by DF or MiM attacks based On Faulty PRFs

Protocol	Distance-Fraud	MiM attack
TDB [1]	page 108	page 109
DFKO (Enhanced Kim-Avoine Protocol) [6]	page 110	–
Hancke and Kuhn's [9]	page 112	–
Avoine and Tchamkerten's [2]	page 114	–
Reid's *et al* [14]	page 115	page 116
Swiss Knife [12]	–	page 118

[1] In the concrete presentation, we will make this clear. Whilst a protocol may have not been claimed to be secure against all frauds, it was claimed to be secure against a specific fraud. In our analysis/exemplification, we will show the contrary.

Structure of the paper. The remainder of the paper is structured as follows. In Section 2, we present some reminders about distance-bounding protocols and PRF functions. In Section 3, we give one general construction of PRFs with trapdoors. This construction prompts to distance-fraud and MiM attacks in DB protocols. In Section 4, we present such attacks on the protocols listed in Table 1. We conclude in Section 5.

2 Distance-Bounding Protocols and the PRF Assumption

In this section, we will recall general facts about distance-bounding protocols and basic notions about pseudorandom functions.

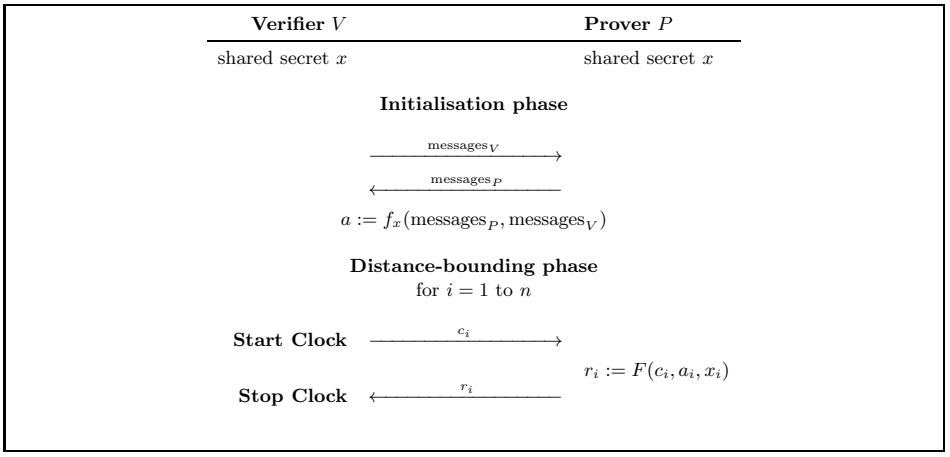

Fig. 1. Informative Sketch on Most Distance-Bounding Protocols

Distance-Bounding Protocols. The great majority of distance-bounding protocols [10,12,14,16] consist of a data-agreement phase or *initialisation phase* and a time-critical, fast computation-based *distance-bounding* phase. Fig. 1 captures the core of distance-bounding (DB). In the initialisation phase, a prover P and a verifier V use their randomnesses, their common secret x and a PRF f to exchange messages_P and messages_V respectively and establish a sub-secret a; a is normally a bitstring or a vector of elements in a finite space of small size. In the DB phase which is time-critical, the responses are normally defined via a response-function F. The i-th (one-bit) response r_i to the i-th randomly picked small-size challenge c_i is most often given by a computation of the sort $F(c_i, a_i, x_i)$, where $i \in \{1, \ldots, n\}$. The initialisation phase makes it possible for both parties to evaluate this function even though they do not have their counterpart's coins (i.e., the two honest parties have agreed over the vector a, they share x_i and they both know c_i).

Pseudorandom functions. A pseudorandom function (PRF) is a family of (polynomially computable) functions: a set of functions of arbitrary-length input and arbitrary-length output indexed on a set of keys. On this family, a computational assumption is taken, which is denoted as the *pseudorandom function (PRF) assumption*, i.e.,: for an instance sampled uniformly from the family, there exists no polynomial algorithm that distinguishes this instance from a real random function based on a black-box interaction with an oracle simulating them.

One can use the game-methodology [15] to formalise the PRF assumption. To this end, we give the descriptions/definitions below. Let $\mathcal{F}$ be a family of functions with domain D and range R. Let b be a bit. Let $\mathcal{D}$ be a ppt. distinguisher that can interact in a black-box manner with an oracle $\mathcal{O}$. We denote this interaction as $\mathcal{D}^{\mathcal{O}}$ and it is depicted as follows in Fig. 2.

1: **Parameters:** security parameter s; *poly* a polynomial; a ppt. algorithm $\mathcal{D}$; $\ell := \ell(s)$; $L := L(s)$; $D = \{0,1\}^{\ell}$; $R = \{0,1\}^{L}$;

2: $view_{\mathcal{D}} := \emptyset$
3: **while** nb. of iterations $\leq poly(s)$ **do**
4: $x \leftarrow \mathcal{D}(view_D; r_{\mathcal{D}})$; $x \in D$
5: if $x =$ "end : b" with $b \in \{0,1\}$, stop and return b
6: $y \leftarrow \mathcal{O}(x)$; $y \in R$
7: $view_{\mathcal{D}} := view_{\mathcal{D}} \cup \{y\}$
8: **end while**
9: return 0

Fig. 2. The $\mathcal{D}^{\mathcal{O}}$ Interaction

Below, we will simply refer to the oracle implementing f^0 or f^1 by f^b accordingly, responding with $f_b(x) \in R$ for a query $x \in D$. Assume the following description of the PRF game, in Fig. 2.

1: **Parameters:** security parameter s; $\ell := \ell(s)$; $L := L(s)$; $D = \{0,1\}^{\ell}, R = \{0,1\}^{L}$; a family $\mathcal{F} := \mathcal{F}(s)$ of functions from $D \rightarrow R$; a ppt. algorithm $\mathcal{D}$; a bit b.

2: $f^0 \longleftarrow_U [D \rightarrow R]$ // pick a random function from D to R
3: $f^1 \longleftarrow_U \mathcal{F}$ //sample a function from the family
4: $\overline{b} \longleftarrow \mathcal{D}^{f^b}$
5: return $\overline{b}$

Fig. 3. The PRF Game $PRF^b_{\mathcal{F},\mathcal{D}}$

The output of the above game (0 or 1) is denoted $Out(PRF^b_{\mathcal{F},\mathcal{D}})$.

Definition 1 (The PRF assumption[2]). *Let s be a security parameter, k, ℓ, L be some parameters taken as functions of s, $\mathcal{K} = \{0,1\}^k$, $\mathcal{D} = \{0,1\}^{\ell}$,*

[2] This is formalised similarly to [13].

$R = \{0,1\}^L$. *Let* $\mathcal{F}$ *be a family of functions* $(f_k)_{k\in\mathcal{K}}$ *with* $f_k : D \to R$[3] *(an indexed-set of functions over* $\mathcal{K}$*).*

We say that the family $\mathcal{F}$ *is a* PRF *or that the family* $\mathcal{F}$ *respects the* PRF assumption *if for any ppt. algorithm* $\mathcal{D}$*,*

$$\left|\Pr[Out(PRF^0_{\mathcal{F},\mathcal{D}}) = 1] - \Pr[Out(PRF^1_{\mathcal{F},\mathcal{D}}) = 1]\right| < negl(s),$$

where negl is a function over natural numbers eventually lower than the inverse of any polynomial and the probability is taken over the random coins of $\mathcal{D}$*.*

We will also employ the notion of a hard-core function.

Definition 2 (Hard-core function). *Let* s *be a security parameter,* k*,* ℓ*,* L *be some parameters taken as functions of* s*,* $\mathcal{K} = \{0,1\}^k$*,* $\mathcal{D} = \{0,1\}^\ell$*,* $R = \{0,1\}^L$*. Let* $\mathcal{F}$ *be a family of functions* $(f_k)_{k\in\mathcal{K}}$ *with* $f_k : D \to R$*. A function* h *on* $\{0,1\}^*$ *is a hard-core function for* $\mathcal{F}$ *if for all polynomial-time oracle adversary* $\mathcal{A}$ *playing the following game, the probability that it wins is negligible.*

1: *pick* $k \in \mathcal{K}$
2: *run* $z = \mathcal{A}^{f_k(\cdot)}$
3: *win if and only if* $z = h(k)$

Note that if $\mathcal{F}$ is a PRF, then the identity function is hard-core. Further, observe that if h truncates to half of the first bits, it may not be hard-core for a PRF. Indeed, let $f_{k_0,k_1}(x) = k_0$ when $x = 0$ and $f_{k_0,k_1}(x) = g_{k_1}(x)$ when $x \neq 0$; if g is a PRF, then f is a PRF as well, but $h(k_0, k_1) = k_0$ is clearly not hard-core. We could still transform a PRF g into a PRF f for which h is hard-core, for instance with $f_{k_0,k_1}(b, x) = g_{k_b}(x)$.

3 PRFs with a Trapdoor

In this section, we are going to show how, out of a PRF $\mathcal{G}$, one can program another PRF $\mathcal{F}$ to accommodate a trapdoor making its instances leak a special value when called on that trapdoor. Otherwise, an instance of the thus-wise constructed PRF $\mathcal{F}$ "behaves" like the corresponding instance of $\mathcal{G}$. The ultimate goal of these constructions is to (help) show that the PRF assumption is not enough for the security of DB protocol, as claimed [1,6]. In fact, inappropriate PRFs used in DB protocols can lead to frauds: the first construction points to distance-fraud and the second to man-in-the-middle (MiM) attacks.

Consider the following informal explanations related to construction. Consider a function σ, with the aim of mapping an element of a domain $\mathcal{K} \times D$ onto an element of a domain R. Typically, σ embeds the input $k \in \mathcal{K}$ so that its output leaks k. Similarly, $\mathsf{correctPad}$ maps elements from $\mathcal{K}$ onto disjoint subsets of the set D above. Also, $\mathsf{correctPad}(k)$ must be such that its inverse is computable,

[3] We denote a function $f_k \in \mathcal{F}$, for a fixed $k \in \mathcal{K}$ as a PRF instance.

i.e., the token k is extractable out of any $\mathsf{correctPad}(k)$ element. We formalise this below and use it to formulate our result on PRF-constructions.

Theorem 3. *Let s be a security parameter. Let the following sizes of domains be expressed in function of s: ℓ, $\bar{\ell}$, L, $\bar{L}$, k. Consider the following three sets $D = \{0,1\}^{\ell}$, $\bar{D} = \{0,1\}^{\bar{\ell}}$, $R = \{0,1\}^{L}$, $\bar{R} = \{0,1\}^{\bar{L}}$, $\mathcal{K} = \{0,1\}^{k}$.*

Let h be a polynomially computable function on $\{0,1\}^*$.

Let $\mathcal{G}$ be a family of functions $(g_k)_{k\in\mathcal{K}}$ and $g_k : \bar{D} \to \bar{R}$. We assume that $\mathcal{G}$ is a PRF and that h is a hard core function for $\mathcal{G}$. Let $T^{\mathcal{O}}$ be a polynomial-time oracle-algorithm accessing $\mathcal{O}$, admitting inputs in D and outputs in R.

Consider a polynomially computable function σ from $\mathcal{K} \times D$ to R.

Consider a map $\mathsf{correctPad}$ from $\mathcal{K}$ to the set of subsets of D such that there exists a polynomial time oracle-algorithm $\mathsf{extract}^{g_k(\cdot)}$ from D such that for any $k \in \mathcal{K}$ and $x \in \mathsf{correctPad}(k)$, we have $\mathsf{extract}^{g_k(\cdot)}(x) = h(k)$. It is further assumed that given x and k, it can be decided in polynomial time whether x belongs to $\mathsf{correctPad}(k)$ or not.

Let a $\mathcal{F}$ be a family of functions $(f_k)_{k\in\mathcal{K}}$ and, for some arbitrarily fixed $k \in \mathcal{K}$, $f_k : D \to R$ defined as follows:

$$f_k(x) = \begin{cases} \sigma(k,x), & \text{if } x \in \mathsf{correctPad}(k) \\ T^{g_k(\cdot)}(x), & \text{otherwise .} \end{cases}$$

Then, the family $\mathcal{F}$ is a PRF.

The proof of Theorem 3 is natural, following the game-reduction methodology [15], by indistinguishability between games based on failure-events.

Proof. We first observe that since membership of $\mathsf{correctPad}$ and σ can be computed in polynomial time, then f is polynomially computable as well.

Let $k \in \mathcal{K}$ be arbitrarily fixed. Consider the distinguisher $\mathcal{D}$ distinguishing $(f_k)_{k\in\mathcal{K}}$ in the $PRF^b_{\mathcal{F},\mathcal{D}}$ game. Let $(x_1, f_k(x_1)), \ldots, (x_n, f_k(x_n))$ be the query-reply tuples between $\mathcal{D}$ and the oracle in $PRF^1_{\mathcal{F},\mathcal{D}}$, for $n \leq poly(s)$, with $poly$ and s defined in $PRF^b_{\mathcal{F},\mathcal{D}}$, $x_i \in D$, $f_k(x_i) \in R$, for all $i \in \{1, \ldots, n\}$.

Clearly, $\Pr[\mathcal{D} \text{ wins in } PRF^0_{\mathcal{F},\mathcal{D}}] = \Pr[\mathcal{D} \text{ wins in } PRF^0_{\mathcal{G},\mathcal{D}}]$. Since $\mathcal{G}$ is a PRF, we further have

$$\left| \Pr[\mathcal{D} \text{ wins in } PRF^0_{\mathcal{G},\mathcal{D}}] - \Pr[\mathcal{D} \text{ wins in } PRF^1_{\mathcal{G},\mathcal{D}}] \right| = negl(s).$$

So, we just have to show that

$$\left| Pr[\mathcal{D} \text{ wins in } PRF^1_{\mathcal{F},\mathcal{D}}] - \Pr[\mathcal{D} \text{ wins in } PRF^1_{\mathcal{G},\mathcal{D}}] \right| = negl(s).$$

Unless $\mathcal{D}$ queries x_i ($i \in \{1, \ldots, n\}$) with $x_i \in \mathsf{correctPad}(k)$, his view is that of $\mathcal{D}$ in the "corresponding" $PRF^1_{\mathcal{G},\mathcal{D}}$ with the same random coins, i.e.,

$$\ldots, (x_{i-1}, T^{g_k(\cdot)}(x_{i-1})), (x_i, T^{g_k(\cdot)}(x_i)), (x_{i+1}, T^{g_k(\cdot)}(x_{i+1})), \ldots, (x_n, T^{g_k(\cdot)}(x_n))$$

In the contrary case, where he does query $x_i \in \mathsf{correctPad}(k)$, the view of $\mathcal{D}$ contains $\sigma(k, x_i)$ instead of $T^{g(x_i)}$ (for this fixed i).

So, the game $PRF^1_{\mathcal{F},\mathcal{D}}$ is indistinguishable from the game $PRF^1_{\mathcal{G},\mathcal{D}}$ unless the failure-event F of querying the specific $x_i \in \mathsf{correctPad}(k)$ occurs. In other words, $\Pr[\mathcal{D} \text{ wins in } PRF^1_{\mathcal{F},\mathcal{D}} | \neg F] = \Pr[\mathcal{D} \text{ wins in } PRF^1_{\mathcal{G},\mathcal{D}} | \neg F]$. So,

$$\left| Pr[\mathcal{D} \text{ wins in } PRF^1_{\mathcal{F},\mathcal{D}}] - \Pr[\mathcal{D} \text{ wins in } PRF^1_{\mathcal{G},\mathcal{D}}] \right| \leq \Pr[F].$$

What is left to be proven is that $\Pr[F]$ is negligible. To bound the probability $\Pr[F]$ of F occurring, we let p_i be the probability that $x_i \in \mathsf{correctPad}(k)$ and that $x_j \notin \mathsf{correctPad}(k)$ for $j \in \{1, \ldots, i-1\}$. Clearly, $\Pr[F] \leq \sum_i p_i$.

So, this reduces to proving that p_i is negligible for each i. To do so, we construct a new algorithm $\mathcal{A}^{g_k(\cdot)}$. Namely, $\mathcal{A}$ simulates $\mathcal{D}$ and T until it computes x_i. Then, the algorithm $\mathcal{A}$ uses x_i to get $k' = \mathsf{extract}^{g_k(\cdot)}(x_i)$.
In the case that $x_j \notin \mathsf{correctPad}(k)$ for $j = 1, \ldots, i-1$, the simulation is perfect. If $x_i \in \mathsf{correctPad}(k)$, then $k' = h(k)$ and we obtain that $\mathcal{A}$ outputs $h(k)$. So, $\Pr[\mathcal{A} \text{ yields } h(k)] \geq p_i$. Since h is hard-core, p_i is negligible. □

The first note on Theorem 3 is that a PRF can be constructed, if PRFs exist. I.e., starting from $\mathcal{G}$ being some PRF, Theorem 3 gives the concrete construction of another PRF $\mathcal{F}$, with a trapdoor.

Then, one of the aims of this result is to indicate that if an inappropriate PRF $\mathcal{F}$ is used in (the initialisation phase of) DB protocols, then a distance-fraud can be mounted onto those protocols. To see this easily, you may want to refresh the notations in Fig. 1 informally describing the DB protocols. Now, imagine a dishonest prover P^* (who of course has the shared-key x and) that wants to mount a distance-fraud onto a DB protocol using a trapdoor-enhanced PRF $\mathcal{F}$ as the one in Theorem 3. By applying an input from $\mathsf{correctPad}(x)$, he sends messages to the verifier V such that $(\text{messages}_P, \text{messages}_V) \in \mathsf{correctPad}(x)$. Then, $f_x(\text{messages}_P, \text{messages}_V) = \sigma(x, \text{messages}_P, \text{messages}_V)$. Usually, in DB protocols (e.g., [1,6,9,2], etc.), messages_P is in fact a nonce N_P and messages_V is a nonce N_V. So, an example of such adaptive choices and exploitation of poor PRFs is the following: P^* can choose adaptively N_P to be, say, x and then $f_x(\text{messages}_P, \text{messages}_V)$ becomes $f_x(x, N_V)$ which is equal –by the trapdoor property– with, say, $x\|x\|\ldots\|x$. Since the responses are based on this output and x, this usually enables P^* to answer any challenge before they even arrive at him. This means that he successfully mounts a distance-fraud attack. Of course, this sort of artificial function and its trapdoor depend on the protocol under discussion, as Section 4 will show. I.e., we need appropriate special $\sigma(x, N_P, N_V)$ to be output of the PRF instances. (Usually, it simply implies that the response is a constant in terms of the challenge). Also, other forms of output of the PRF instances can be imagined, as long as they facilitate the responses of the DB phase to be independent from the challenge (i.e., instead of $\sigma(x, \text{messages}_P, \text{messages}_P)$ we could directly some constant *cte* known to P^* and lying in the appropriate domains used in the above theorem). It is also needed that the distribution of such outputs and the domains we have at hand, $\sigma(x, \text{messages}_P, \text{messages}_P)$ seems a reasonable choice for the "conned" protocols participants.

This construction of a "trapdoor PRF" (from a given PRF) uses an oracle in the inversion of correctPad only for the purpose of it giving raise not only to DF but to MiM attacks also. The basic idea of MiM attacks of this sort relies on a PRF $(f_x)_{x\in\mathcal{K}}$ such that $f_x(y) = x$ when $y = g_x(cte) + x$, where g_x is a PRF instance from a given PRF $\mathcal{G}$. By adapting this generic construction, we could have an adversary first getting y by querying a specific set of challenges c_i to the prover, then using y as a nonce to extract x from the prover. A specific, detailed description of an attack of the sort is presented in page 109 against the TDB protocol [1]. For the proof of such $(f_x)_{x\in\mathcal{K}}$ being a PRF when constructed as in Theorem 3, there is the need that the inversion of correctPad is made via an access to an oracle of the stated sort.

4 PRF-Based Attacks

4.1 TDB Protocol

In this protocol, due to [1] and depicted in Fig. 4, the prover P and the verifier V share a secret s that can be viewed as a vector $(s_1, \ldots, s_m)$ of m coordinates over a group G, i.e., $s_i \in G$, $i \in \{1, m\}$. The prover P and the verifier V use an (n, k) threshold scheme on some sub-secrets obtained via a pseudo-random function instance f_s. Like in most cases, the protocol is divided into two phases: the *initialisation* phase and the *distance-bounding* phase.

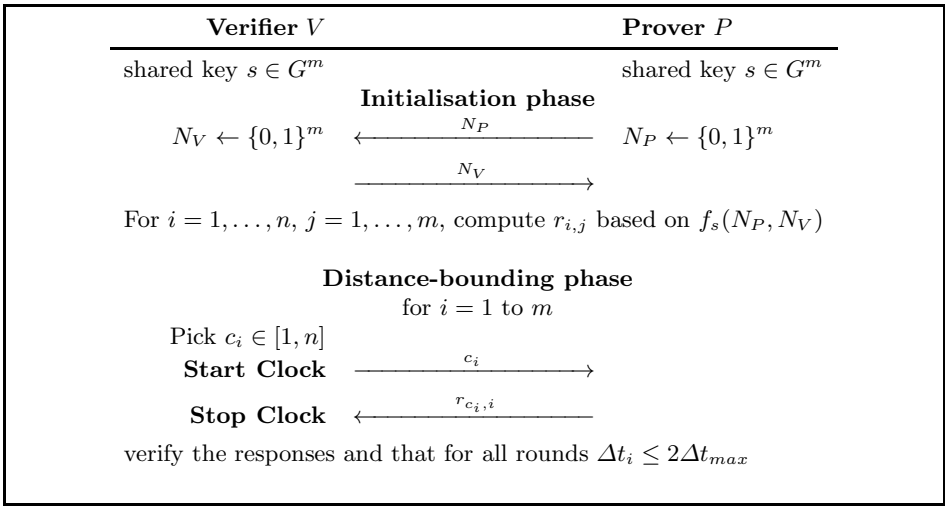

Fig. 4. The TDB protocol [1]

- **Initialisation Phase:** This phase is not time critical. The prover P and the verifier V select two random nonces N_P and N_V correspondingly and transmit them to each other. Then, both the prover P and the verifier V compute an $n \times m$ matrix $\mathcal{R}$, where each column $(r_{1,i}, r_{2,i}, \ldots, r_{n,i})^T$ of $\mathcal{R}$ is obtained using the

(n, k) threshold scheme applied on s_i. Namely, the $r_{c_i,i}$'s are generated with the help of a pseudorandom function f by computing $f_s(N_P, N_V)$ and the elements of the last row of the matrix are generated by summing all the elements in the upper rows on the same column and adding the corresponding secret bit of s. For instance, if we consider the case where $n = k = 3$, $G = F_2$ and a specific secret sharing scheme, then the response matrix will have dimension $3 \times m$ and will have the form:

$$\mathcal{R}_1 = \begin{pmatrix} r_{1,1} & \cdots & r_{1,m} \\ r_{2,1} & \ddots & r_{2,m} \\ s_1 \oplus r_{1,1} \oplus r_{2,1} & \cdots & s_m \oplus r_{1,m} \oplus r_{2,m} \end{pmatrix}$$

- **Distance Bounding Phase:** This phase is time critical and involves the exchange of challenges-responses (rounds) at maximum bit-rate. Such an exchange is repeated m times (i.e., there are m rounds). Assume a generic round i (for i varying from 1 to m). At each round i the challenge-response delay Δt_i is measured. The verifier V starts by choosing a random c_i in its domain $\{1, \ldots, n\}$, initialising the clock to zero and transmitting c_i to P. The prover P responds with $r_{c_i,i}$ which denotes the element located at the c_i-th row and the i-th column of the table $\mathcal{R}$. On receiving $r_{c_i,i}$, V stops the clock and stores the received response and the delay time Δt_i.

After the end of the distance-bounding phase, a verification phase is performed and the verifier V checks if the received responses are correct and if for the response times Δt_i it holds that $\Delta t_i \leq 2\Delta t_{max}$, where Δt_{max} denotes the time it takes for a bit to be transmitted from the prover to the verifier.

Based on the construction and significance of Theorem 3, we construct the following attack on an instantiation of the TDB protocol. The same sort of attack would work for other instantiations of the TDB protocol (i.e., with different secret sharing schemes inside).

Distance Fraud Attack on an Instantiation of the TDB Protocol. Let g be a PRF from $\{0,1\}^{2m}$ to itself. Let us consider the PRF f constructed from Theorem 3 based on g and the following elements. Let $T^g(x) = g(x)$. Let D be the set of (N_P, N_V) pairs. Let $\sigma(s, N_P, N_V) = s \| s$, $\mathsf{correctPad}(s) = \{s \| N_V; N_V \in \{0,1\}^m\}$, and $\mathsf{extract}^{g_s(.)}(N_P, N_V) = N_P$. We have

$$f_s(N_P, N_V) = \begin{cases} s \| s, & \text{if } N_P = s \\ g_s(N_P, N_V), & \text{otherwise} \end{cases}$$

By Theorem 3, f is a PRF. Consider an instantiation of the TDB protocol, where the response matrix is $\mathcal{R}_1$ above and the PRF f is being used.

In this instance of the TDB protocol, it is obvious that a legitimate, far-away but dishonest prover could easily perform a distance-fraud attack. He just needs to choose N_P to be equal to s (as shown above). Then, the $\mathcal{R}_1$ matrix has all its rows equal to s. So for any challenge c_i the response will be the i-th bit of the secret key s. This sort of fixed responses can be sent before receiving the

challenge. Thus, he can defeat the distance-bound. The extension to n, k greater than 3 is trivial: in the trapdoor case of f_s, one repeats s for $n-1$ times and then considers the case where n is odd and even separately.

So, if a PRF exists, then we can exhibit instances of TDB which are insecure against DF! The PRF assumption is not enough for the security of the TDB protocols against DF.

Man-in-the-Middle Attack on an Instantiation of the TDB Protocol. Consider again the instantiation of the TDB protocol with $n = k = 3$, $G = F_2$ and with $\mathcal{R}_1$ being the response matrix. Let the shared key be denoted by s. Let g be a PRF mapping $\{0,1\}^{\frac{m}{2}}$ to itself and from $\{0,1\}^{2m}$ to $\{0,1\}^{\frac{3m}{2}}$. We assume that the least significant half of s is hard-core for g. We define $T^{g_s(\cdot)}(N_P, N_V) = (\alpha, \beta, \gamma, \beta \oplus g_s(\alpha))$ where $g_s(N_P, N_V) = (\alpha, \beta, \gamma)$. Let us consider the following elements: $\sigma(s, N_P, N_V) = s \| s$, $\mathsf{correctPad}(s) = \{N_P \| \overline{\alpha} \| (g_s(\overline{\alpha}) \oplus \mathsf{lsb}_{\frac{m}{2}}(s)) \, ; \, \overline{\alpha} \in \{0,1\}^{\frac{m}{2}}\}$, and $\mathsf{extract}^{g_s(\cdot)}(N_P, \overline{\alpha}, \overline{\beta}) = \overline{\beta} \oplus g_s(\overline{\alpha})$. Let f be constructed from Theorem 3 based on g as below. By Theorem 3, f is a PRF.

$$f_s(N_P, N_V) = \begin{cases} (\alpha, \beta, \gamma, \beta \oplus g_s(\alpha)), & \text{if } N_V \text{ is not of the form } \overline{\alpha} \| (g_s(\overline{\alpha}) \oplus \mathsf{lsb}_{\frac{m}{2}}(s)) \\ & \text{and } (\alpha, \beta, \gamma) = g_s(N_P, N_V) \\ s \| s, & \text{if } N_V = \overline{\alpha} \| (g_s(\overline{\alpha}) \oplus \mathsf{lsb}_{\frac{m}{2}}(s)), \text{ for some } \overline{\alpha} \end{cases}$$

In the notations of the TDB protocol, $r_1 = (\alpha, \beta)$ and $r_2 = (\gamma, \beta \oplus g_s(\alpha))$.

We are now going to explain the attack. The attacker has the goal of recovering s from the prover, so that he can later impersonate this prover as he pleases. To do so, the attacker impersonates first the verifier to the prover. He sends an arbitrary N_V, so the prover calculates the generic subsecret vectors $(\alpha', \beta', \gamma', \psi')$ as some $(\alpha, \beta, \gamma, \beta \oplus g_s(\alpha))$. Then the adversary sends many challenges equal to 1, $c_i = 1$, e.g., for $i \in \{1, \ldots, \frac{m}{2}\}$. In this way, he gets the first half of the first subsecret-vector $r_1{=}(\alpha, \beta)$, i.e., he obtains α,. Then, the adversary sends the prover many challenges equal to 3, some $c_i = 3$. By the secret sharing scheme used, the responses to the latest challenges are equal to $r_1 + r_2 + s = (\alpha, \beta) \oplus (\gamma, \beta \oplus g_s(\alpha)) \oplus s = (\alpha \oplus \gamma, g_s(\alpha)) \oplus s$. So, from this approach, the attacker gets $g_s(\alpha) \oplus \mathsf{lsb}_{\frac{m}{2}}(s)$. Finally, he can now form $N'_V = \alpha \| (g_s(\alpha) \oplus \mathsf{lsb}_{\frac{m}{2}}(s))$. The second step of the attack (in a new hijacked session in which the attacker is again impersonating the verifier to the honest prover) consists in the attacker to employ his knowledge gained as above to choose N_V equal to N'_V. By then injecting any challenges to the prover, the attacker will know (due to the built-in PRF) that the responses of the prover will be the bits of s. Like this, he will learn the whole of the secret key and he will be subsequently able to impersonate this prover in any circumstance.

Again, according to Theorem 3, the resulting family of functions $(f_s)_{s \in G^m}$ is a PRF. The attack exhibited does therefore disprove the claims of MiM security in [1] based solely on the PRF assumption.

4.2 DFKO Protocol (Enhancement of the Kim-Avoine Protocol)

The protocol proposed by Dürholz *et al.* [6] is based on the protocol proposed by Kim and Avoine [11]. More precisely, the protocol proposed by Kim and Avoine [11] is claimed to be mafia and distance-fraud resistant. Dürholz *et al.* [6] have modified it in order to safeguard against impersonation attacks. Its sketch is depicted in Fig. 5. This protocol is again discriminated into two phases: the *initialisation phase* and the *distance-bounding phase*. During the *initialisation phase* both the prover P and the verifier V select a random nonce N_P and a random nonce N_V correspondingly. Then, both of them compute the output of a PRF instance f_x that takes as input the two random nonces that they have exchanged, i.e.,: $I\|C\|D\|v^0\|v^1 := f_x(N_V\|N_P)$, where x is shared secret/key. The prover P also sends part of this output (i.e., I) to the verifier and V verifies that this is the correct value for I.

During the *distance-bounding phase* the verifier V sends the bit-challenges R_i to the prover that are either generated randomly or using the value D (i.e., a part of the output of $f_x(N_P, N_V)$ generated in the *initialisation phase*). If the challenges are random, then the responses are also random otherwise the responses depend on the value v^0 (i.e., a part of the output of $f_x(N_P, N_V)$ generated in the *initialisation phase*). For more details regarding the Dürholz *et al.* protocol, we refer the readers to [6].

Based on Theorem 3, we now give an attack on an instantiation of DFKO.

Distance Fraud Attack on the DFKO Protocol. Let g be a PRF with parameters as the one needed in the DFKO protocol. Let us consider the PRF f constructed from Theorem 3 based on g and the following elements. Let $T^g(x) = g(x)$. Let $\sigma(x, N_P, N_V) = I\|0\cdots0\|D\|v^0\|v^1$ where $g_x(N_P, N_V)$ parses into $I\| \cdot \|D\|v^0\|v^1$, $\mathsf{correctPad}(x) = \{x\|N_V; N_V \text{ nonce}\}$, and $\mathsf{extract}^{g_x(\cdot)}(N_P, N_V) = N_P$. We have

$$f_x(N_P, N_V) = \begin{cases} I\|0\ldots0\|D\|v^0\|v^1, & \text{if } N_P = x \text{ for } g_x(x, N_V) = I\|C\|D\|v^0\|v^1 \\ g_x(N_P, N_V) & \text{otherwise,} \end{cases}$$

By Theorem 3, f is a PRF.

Consider an instantiation of the DFKO protocol, where the PRF f is being used. In this instance of the DFKO protocol, if a prover is dishonest and picks N_P to be equal to x, then the response of the prover P will always be a bit of D. Thus, this dishonest prover would be able to know in advance all the responses to every possible challenge. Thus, when being in fact far-away from V, he would perform successfully a distance-fraud attack. This obviously contradicts the security against distance-fraud attacks that is claimed by Dürholz *et al.* in Theorem 2 of [6], based solely on the PRF assumption and described only by a sketch-proof in the appendix of [6].

4.3 Hancke and Kuhn's Protocol

Hancke and Kuhn's protocol [9] is again separated conceptually into two phases: an *initialisation phase* and a *distance-bounding phase*. In the *initialisation phase*

Verifier V		**Prover** P
shared key x		shared key x
	Initialisation phase	
$N_V \leftarrow \{0,1\}^*$	$\xrightarrow{N_V}$	$N_P \leftarrow \{0,1\}^*$
$I\|C\|D\|v^0\|v^1 := f_x(N_V\|N_P)$	$\xleftarrow{N_P, I}$	$I\|C\|D\|v^0\|v^1 := f_x(N_V\|N_P)$
Verify I		
set $cnt := 0$; $err_V := 0$		set state $st = \perp$; $err_T := 0$
	Distance-bounding phase	
	for $i = 1$ to n	
Pick $S_i \in_U \{0,1\}$		
$R_i := S_i$, if $C_i = 1$		
$R_i := D_i$, if $C_i = 0$		
Start Clock	$\xrightarrow{R_i}$	
		if $st \neq rnd$ do:
		if $C_i = 1$, then $T_i = v_i^0$ if $R_i = 0$
		$T_i = v_i^1$ if $R_i = 1$
		if $C_i = 0$, then $T_i = v_i^0$ if $R_i = D_i$
		$T_i \in_U \{0,1\}$ if $R_i \neq D_i$.
		if $R_i \neq D_i$, do $err_T := err_T + 1$
		if $err_T > E_{max}$, do $st = rnd$
		else $T_i \in_U \{0,1\}$
Stop Clock	$\xleftarrow{T_i}$	
set $err_V := err_V + 1$ if T_i does not match		
set $cnt := cnt + 1$ if $\Delta t > t_{max}$		
verify the responses and that for all rounds $\Delta t_i \leq 2\Delta t_{max}$		
output $b = 1$ if $cnt \leq T_{max}$ and$err_R \leq E_{max}$, else $b = 0$		

Fig. 5. Protocol Dürholz *et al.* [6] (Enhanced Kim-Avoine Protocol)

the verifier V and the prover P exchange nonces N_V and N_P correspondingly, then both of them compute the output of a PRF instance f_x that takes as input the two nonces, i.e., $v^0 \| v^1 := f_x(N_P, N_V)$, where x is the shared secret/key. During the *distance-bounding phase* the verifier V selects a random bit-challenge c_i, where $i \in \{1, \ldots, n\}$ and the prover P responds with a bit r_i that has the following form: $r_i := \begin{cases} v_i^0, & \text{if } c_i = 0 \\ v_i^1, & \text{if } c_i = 1. \end{cases}$
After the end of the distance-bounding phase the verifier checks all the received responses and if the response times Δt_i satisfy the condition: $\Delta t_i \leq 2\Delta t_{max}$. The protocol is depicted in Fig. 6.

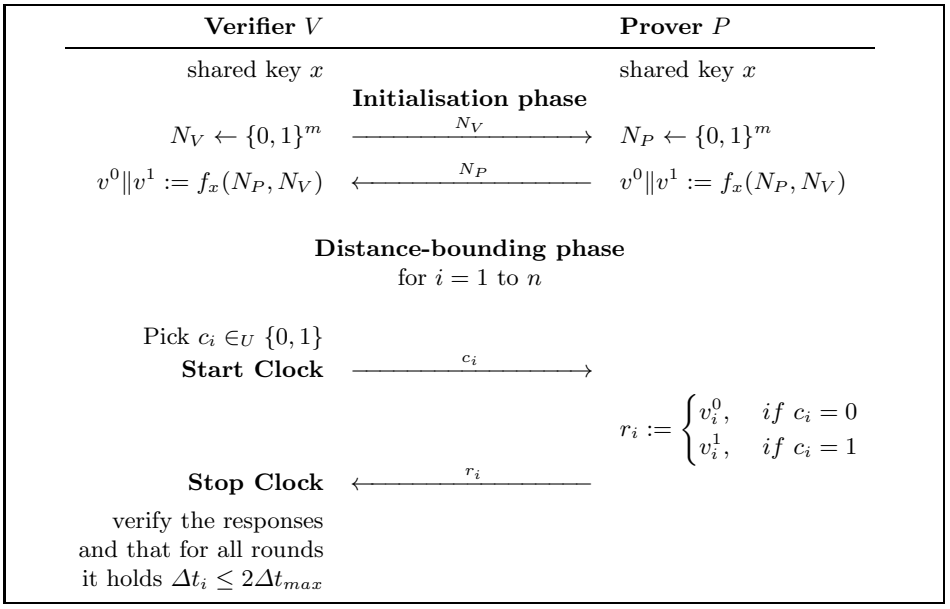

Fig. 6. Protocol proposed by Hancke and Kuhn's [9]

Based on the construction and significance of Theorem 3, we construct the following attack on an instantiation of Hancke and Kuhn's protocol.

Distance Fraud Attack on Hancke and Kuhn's Protocol. Let g be a PRF with parameters as the one needed in the protocol. Let us consider the PRF f constructed from Theorem 3 based on g and the following elements. Let $T^g(x) = g(x)$. Let D be the set of (N_P, N_V) pairs. Let $\sigma(x, N_P, N_V) = v \| v$ where v is an arbitrary constant, $\mathsf{correctPad}(x) = \{x \| N_V; N_V \in \{0,1\}^m\}$, and $\mathsf{extract}^{g_x(.)}(N_P, N_V) = N_P$. We have

$$f_x(N_P, N_V) = \begin{cases} v \| v, & \text{if } N_P = x \\ g_x(N_P, N_V), & \text{otherwise} \end{cases}$$

By Theorem 3, f is a PRF.

Consider an instantiation of the Hancke and Kuhn where the PRF f is being used. In this instance of the Hancke and Kuhn protocol, if a prover is dishonest and picks N_P to be equal to x, then the responses r_i will always be the same for all challenges c_i where $i \in \{1, \ldots, n\}$. Thus, such a legitimate, far-away but dishonest prover can successfully perform a distance-fraud attack by sending the responses before receiving the challenges. This obviously contradicts the security against distance-fraud attacks that was claimed in Theorem 3.2 of [7] solely based on the PRF assumption and that claims of [9].

4.4 Avoine and Tchamkerten's Protocol

This protocol from [2], presented in Fig. 7, is again divided into two phases an *initialisation* and a *distance-bounding* base. The prover P and the verifier V share a common secret x and they have agreed on some parameters m and n. In the *initialisation phase* which is not time critical, the verifier V selects a random nonce N_V and transmits it to the prover P. The prover P also selects a random nonce N_P and transmits it to V. Then, they compute the output of a PRF instance f_x on the input given by the two nonces N_P and N_V, i.e., $v^0 \| v^1 := f_x(N_P, N_V)$. The output of this computation has length at least $m + 2^{n+1} - 2$. We denote the first m bits of this output by v^0 and the rest of the bits by v^1. Then, the prover P sends to the verifier V the value v^0 for verification purposes.

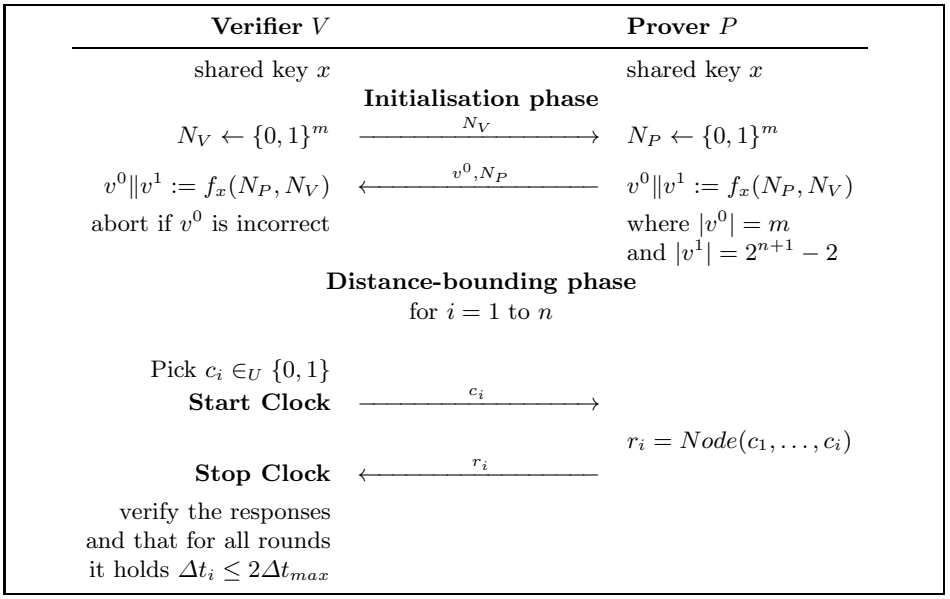

Fig. 7. Protocol proposed by Avoine and Tchamkerten [2]

For the *distance-bounding phase* using v^1, the prover P and the verifier V label a full binary tree of depth n. The left and right edges of the tree are

labelled with 0 and 1 respectively, while each node of the tree (except of the root node) is assigned a value of a particular bit of v^1 in an one-to-one fashion (starting from the lowest level nodes and moving from left to right, moving up the tree after assigning all the nodes in the current level). The *distance-bounding* phase has n challenge-response exchanges/rounds. At each round i the challenge-response delay Δt_i is measured. The verifier V chooses a random bit c_i, initialises the clock to zero and transmits c_i to P. Then, P answers with the value $r_i = Node(c_1, \ldots, c_i)$. This function gives the value of the label of node in the tree which we would reach from the root by taking the path labelled $c_1, c_2, \ldots, c_i$ on the edges.

Based on the construction and significance of Theorem 3, we construct the following attack on an instantiation of the Avoine and Tchamkerten protocol.

Distance Fraud Attack on the Avoine and Tchamkerten Protocol. Let g be a PRF with parameters as the one needed in the protocol. Let us consider the PRF f constructed from Theorem 3 based on g and the following elements. Let $T^g(x) = g(x)$. Let D be the set of (N_P, N_V) pairs. Let $\sigma(x, N_P, N_V) = v^0 \| T$ where v^0 and T are constant defined below, $\mathsf{correctPad}(x) = \{x \| N_V; N_V \in \{0,1\}^m\}$, and $\mathsf{extract}^{g_x(.)}(N_P, N_V) = N_P$. We have

$$f_x(N_P, N_V) = \begin{cases} v^0 \| T, & \text{if } N_P = x \\ g_x(N_P, N_V), & \text{otherwise} \end{cases}$$

The constant v^0 is an arbitrary admissible value for v^0. The constant T denotes the full binary tree of length n where all the paths constructed by reading-out the labels of the nodes are equal. A tree with such a property is, e.g., the one where all nodes in the same level have the same label (either all 0 or all 1 per one level). By Theorem 3, f is a PRF.

Consider an instantiation of the Avoine and Tchamkerten protocol, where the PRF f is being used. In that case a dishonest, far-away prover which forges N_P to x will always give the correct response without the need to wait for the challenge to arrive. Thus, he would be able to respond earlier and perform successfully a distance-fraud attack. This obviously contradicts the security against distance-fraud attacks that was claimed in Theorem 3.3 of [7], page 11.

4.5 Reid's *et al.* Protocol

In the Reid *et al.* [14] protocol (depicted in Fig. 8, the prover and the verifier that share a secret key x. During the initialisation phase both of them generate random nonces N_P and N_V and exchange them, as well as exchanging their identities. Then both of them generate a session key k as $k := f_x(ID_P \| ID_V \| N_V \| N_P)$ and encrypt the shared key x with the session key k, i.e., $e := \mathcal{E}_k(x)$, where f_x is a PRF instance. One can view k as an ephemeral key. Based on Theorem 3.4 in [7], the assumption needed for the security of this protocol is that $\mathcal{E}$ should be a IND-CPA secure, symmetric encryption. For instance, we can use one-time pad $\mathcal{E}_k(x) = x \oplus k$.

The distance-bounding phase contains n rounds. At each round i the challenge-response delay Δt_i is measured, where $i \in \{1, \ldots, n\}$. The verifier chooses a random challenge c_i and the prover responds with r_i such that:

$$r_i := \begin{cases} e_i, & \text{if } c_i = 0 \\ k_i, & \text{if } c_i = 1 \end{cases}$$

After the end of the distance-bounding phase the verifier checks the responses and verifies that all response times are below a pre-defined threshold.

Based on the construction and significance of Theorem 3, we construct the following attack on an instantiation of Reid *et al.*'s protocol.

Distance-Fraud Attack on Reid's* et al. *Protocol. Let g be a PRF with parameters as the one needed in the protocol. Let us consider the PRF f constructed from Theorem 3 based on g and the following elements. Let $T^g(x) = g(x)$. Let D be the set of $(ID_V \| ID_P \| N_P, N_V)$ tuples. Let $\sigma(x, ID_V, ID_P, N_V, N_P) = x$, $\mathsf{correctPad}(x) = \{ID_V \| ID_P \| N_V \| x\}$, and $\mathsf{extract}^{g_x(\cdot)}(ID_V, ID_P, N_V, N_P) = N_P$. We have

$$f_x(ID_V \| ID_P \| N_V \| N_P) = \begin{cases} x, & \text{if } N_P = x \\ g_x(ID_V \| ID_P \| N_V \| N_P) & \text{otherwise,} \end{cases}$$

By Theorem 3, f is a PRF.

Verifier V		**Prover** P
shared key x		shared key x
Initialisation phase		
$N_V \leftarrow \{0,1\}^m$	$\xrightarrow{ID_V, N_V}$	$N_P \leftarrow \{0,1\}^m$
$k := f_x(ID_V \| ID_P \| N_V \| N_P)$	$\xleftarrow{ID_P, N_P}$	$k := f_x(ID_V \| ID_P \| N_V \| N_P)$
$e := \mathcal{E}_k(x)$		$e =: \mathcal{E}_k(x)$
Distance-bounding phase		
for $i = 1$ to m		
Pick $c_i \in_U \{0,1\}$		
Start Clock	$\xrightarrow{c_i}$	
		$r_i := \begin{cases} e_i, & \text{if } c_i = 0 \\ k_i, & \text{if } c_i = 1 \end{cases}$
Stop Clock	$\xleftarrow{r_i}$	
verify the responses and that for all rounds it holds $\Delta t_i \leq 2\Delta t_{max}$		

Fig. 8. Protocol proposed by Reid *et al.* [14]

Consider an instantiation of the Reid's *et al.* protocol where the PRF f is being used. Also, we assume that the following encryption function instance is employed:

$$\mathcal{E}_k^{new}(x) = \begin{cases} \mathcal{E}_k(x), & \text{if } k \neq x \text{ and } \mathcal{E}_k(x) \neq x \\ x, & \text{if } k = x \\ \mathcal{E}_k(k), & \text{if } \mathcal{E}_k(x) = x \text{ and } k \neq x. \end{cases}$$

Similarly to Theorem 3, we can show that if $\mathcal{E}$ is an IND-CPA secure symmetric encryption, so is $\mathcal{E}^{new}$.

If a far-away dishonest prover indeed chooses N_P to be equal to x, then the responses r_i will always be equal to x_i (for all $i \in \{1, \ldots, n\}$). Thus, a dishonest, far-away prover can perform a successful distance-fraud attack and claim that he is nearer to the verifier than he really is. This obviously contradicts the security against distance-fraud attacks given in Theorem 3.4, in page 13 of [7], solely based on the PRF assumption and Theorem 1, in page 17 of [14].

Another weak PRF leading to a distance-bounding attack is provided in the next example.

Man-in-the-Middle Attack on Reid's* et al.*'s Protocol. We first construct a PRF producing some unforgeable outputs. To this end, we start with a PRF g such that $f_x(u, v) = g_x(u, v) \| g_x(g_x(u, v))$ has parameters as the one needed in the protocol. We define a predicate $V_x(a, b)$ which is true if and only if $g_x(a) = b$. Clearly, $V_x(f_x(u, v))$ holds for all x, u, v. It is easy to see that f is a PRF. Next, we consider the encryption function $\mathcal{E}$ defined by

$$\mathcal{E}_k(y) = \begin{cases} y, & \text{if } V_y(k) \text{ or } V_y(k \oplus y) \\ k \oplus y, & \text{otherwise} \end{cases}$$

We can show that, for k random (an unknown to the adversary), it is hard to forge y such that $V_y(k)$ or $V_y(k \oplus y)$ hold. So, $\mathcal{E}$ if IND-CPA.

Consider now an instantiation of Reid's *et al.* protocol, where f and $\mathcal{E}$ are as constructed. In this instantiation of the protocol, the encryption is such that $\mathcal{E}_k(x) = x$ for all choices of the nonces. The attacker impersonates the verifier to the prover. First, he starts a session in which he inflicts $N_V = 0$. So, in this session, he sends many challenges equal to 0. Like this, he retrieves $\mathcal{E}_k(x)$ from the responses, which is the secret key x.

Note that by changing the encryption so that $\mathcal{E}_k(x) = k$, we can make a distance fraud attack.

4.6 The Swiss-Knife Protocol

In the Swiss-Knife protocol [12] (depicted in Fig. 9), the prover and the verifier share a secret key x. During the initialisation phase both of them generate random nonces N_P and N_V correspondingly and exchange them. Furthermore, both of them generate a session key a as: $a := f_x(cte, N_P)$, where cte denotes a constant and two values Z^0 and Z^1 such that: $\begin{cases} Z^0 := a \\ Z^1 := a \oplus x \end{cases}$

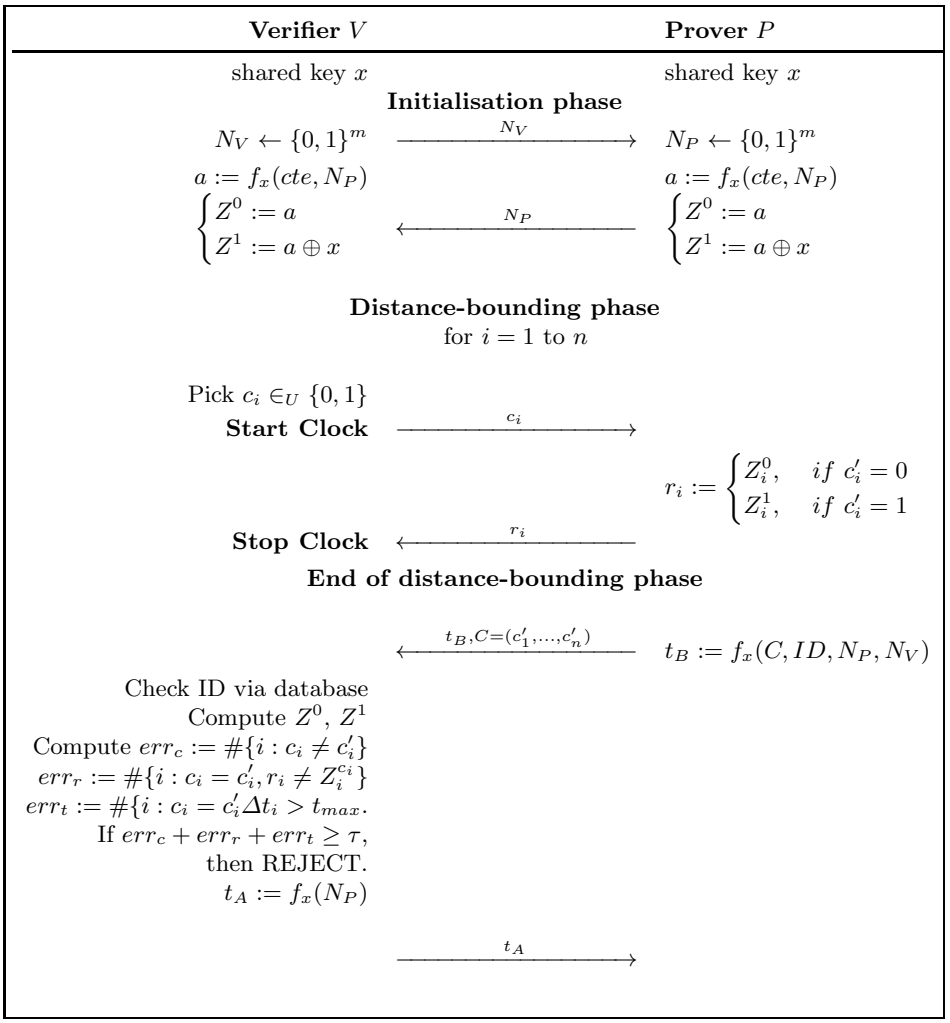

Fig. 9. Swiss-Knife protocol [12]

In the distance bounding phase which is repeated n times the verifier selects a random challenge c_i where $i \in \{1 \dots n\}$ and the prover responds with r_i such that:

$$r_i := \begin{cases} Z_i^0, & if\ c_i' = 0 \\ Z_i^1, & if\ c_i' = 1, \end{cases}$$

where c_i' is the challenge that the prover actually received in the i-th round, i.e., c_i' will be c_i itself, if the transmission was correct, or c_i' will be $\overline{c_i}$, if c_i was perturbed by noise. After the end of the distance bounding phase the prover transmits a message t_B such that: $t_B := f_x(C, ID, N_P, N_V)$ where $C = c_1', \dots, c_n'$.

Based on the construction and significance of Theorem 3, we construct the following attack on an instantiation of the Swiss-Knife protocol.

Man-in-the-Middle Attack on the Swiss-Knife Protocol. Let g be a PRF with parameters as the one needed in the protocol such that the function truncating the the leading half is hard-core. Let us consider the PRF f constructed from Theorem 3 based on g and the following elements. Let $T^g(x) = g(x)$. Let D be the set of (C, ID, N_P, N_V) tuples. Let $\sigma(x, C, ID, N_P, N_V) = x$, $\mathsf{correctPad}(x) = \{C \| ID \| N_P \| N_V; C = 1^{\frac{m}{2}} \| \mathsf{msb}_{\frac{m}{2}}(g_x(cte, N_P) \oplus x)\}$, and $\mathsf{extract}^{g_x(.)}(C, ID, N_P, N_V) = \mathsf{lsb}_{\frac{m}{2}}(C) \oplus \mathsf{msb}_{\frac{m}{2}}(g_x(cte, N_P))$. (Note that extract only recovers the leading half of x so it is not exactly compatible with the assumptions of Theorem 3.) We have

$$f_x(cte, N_P) = g_x(cte, N_P),$$
$$f_x(C, ID, N_P, N_V) = \begin{cases} x, & \text{if } C = 1^{\frac{m}{2}} \| \mathsf{msb}_{\frac{m}{2}}(g_x(cte, N_P) \oplus x) \\ g_x(C, ID, N_P, N_V), & \text{otherwise,} \end{cases}$$
$$f_x(N_P) = g_x(N_P).$$

By Theorem 3, f is a PRF. Indeed, since extract only recovers half of x, we need another trick in the proof of Theorem 3 to show that $\Pr[F]$ is negligible.

Consider an instantiation of the Swiss-Knife protocol where the PRF f is being used. Then, an adversary can extract the key and conduct successfully an impersonation attack. Namely, he can query c_i=1 for $i \in \{1, \dots, \frac{n}{2}\}$ and $c_i = r_{i-\frac{n}{2}}$ for $i = \{\frac{n}{2} + 1, \dots, n\}$. Given the error-tolerance of the protocol, we can presume that the adversary is powerful enough to make the communication noiseless and thus the prover will respond to this very challenges, and, due to the shape of f_x, the adversary will learn x out of his strategy.

The attack exhibited does therefore disprove the claim of MiM security in Theorem 3.5 in [7] based solely on the PRF assumption and the achievement of authentication claimed in [12]. Moreover, it appears [12] that Swiss-Knife was aimed to resist MiM attacks.

5 Conclusions

In this paper, we gave two constructions of PRFs with trapdoors by PRF programming, assuming that PRFs exist. These constructions respectively prompt

to distance-frauds and MiM attacks in DB protocol. In fact, we presented such attacks on important DB protocols, thus disproving different security claims or proofs that appeared in the literature. The latter claims were relying on the PRF assumption for families of function used inside these DB protocol. Our results show that such an assumption is then not enough for the security of DB protocols.

As future work, we will prove how to restore security by additional tricks. Distance fraud security can be achieved by key-masking, i.e., by using $f_x(\cdot) \oplus M$ for a random M instead of $f_x(\cdot)$. MiM security can be restored by introducing an extra security notion to the PRF, so that using $f_x(\cdot) \oplus x$ is still safe.

Acknowledgements. The authors acknowledge the support of the Marie Curie IEF project "PPIDR: Privacy-Preserving Intrusion Detection and Response in Wireless Communications", grant number: 252323, and of the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), under the Swiss National Science Foundation.

References

1. Avoine, G., Lauradoux, C., Martin, B.: How Secret-sharing can Defeat Terrorist Fraud. In: Proceedings of the 4th ACM Conference on Wireless Network Security – WiSec 2011, Hamburg, Germany. ACM, ACM Press (June 2011)
2. Avoine, G., Tchamkerten, A.: An Efficient Distance Bounding RFID Authentication Protocol: Balancing False-Acceptance Rate and Memory Requirement. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 250–261. Springer, Heidelberg (2009)
3. Brands, S., Chaum, D.: Distance Bounding Protocols (Extended Abstract). In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 344–359. Springer, Heidelberg (1994)
4. Cremers, C., Rasmussen, K.B., Čapkun, S.: Distance hijacking attacks on distance bounding protocols. Cryptology ePrint Archive, Report 2011/129 (2011), http://eprint.iacr.org/
5. Drimer, S., Murdoch, S.J.: Keep your enemies close: distance bounding against smartcard relay attacks. In: Proceedings of the 16th USENIX Security Symposium on USENIX Security Symposium, pp. 7:1–7:16. USENIX Association, Berkeley (2007)
6. Dürholz, U., Fischlin, M., Kasper, M., Onete, C.: A Formal Approach to Distance-Bounding RFID Protocols. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 47–62. Springer, Heidelberg (2011)
7. Fischlin, M., Onete, C.: Provably secure distance-bounding: an analysis of prominent protocols. Cryptology ePrint Archive, Report 2012/128 (2012)
8. Ford. Safe and Secure *SecuriCode*™ Keyless Entry (2011), http://www.ford.com/technology/
9. Hancke, G.P., Kuhn, M.G.: An RFID Distance Bounding Protocol. In: Proceedings of SECURECOMM, pp. 67–73 (2005)
10. Kapoor, G., Zhou, W., Piramuthu, S.: Distance Bounding Protocol for Multiple RFID Tag Authentication. In: Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, EUC 2008, vol. 02, pp. 115–120. IEEE, IEEE Computer Society, Shanghai, China (2008)

11. Kim, C.H., Avoine, G.: RFID Distance Bounding Protocol with Mixed Challenges to Prevent Relay Attacks. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 119–133. Springer, Heidelberg (2009)
12. Kim, C.H., Avoine, G., Koeune, F., Standaert, F.-X., Pereira, O.: The Swiss-Knife RFID Distance Bounding Protocol. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 98–115. Springer, Heidelberg (2009)
13. Nielsen, J.B.: A Threshold Pseudorandom Function Construction and Its Applications. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 401–416. Springer, Heidelberg (2002)
14. Reid, J., Gonzalez Nieto, J.M., Tang, T., Senadji, B.: Detecting Relay Attacks with Timing-based Protocols. In: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security, ASIACCS 2007, pp. 204–213. ACM, Singapore (March 2007)
15. Shoup, V.: Sequences of Games: a Tool for Taming Complexity in Security Proofs (2006) (manuscript)
16. Tu, Y.-J., Piramuthu, S.: RFID Distance Bounding Protocols. In: Proceedings of the First International EURASIP Workshop on RFID Technology (2007)

Lattice-Based Hierarchical Inner Product Encryption

Michel Abdalla[1], Angelo De Caro[2], and Karina Mochetti[3]

[1] Département d'Informatique, École Normale Supérieure, France
Michel.Abdalla@ens.fr
http://www.di.ens.fr/users/mabdalla
[2] Dipartimento di Informatica ed Applicazioni, Università degli Studi di Salerno, Italy
decaro@dia.unisa.it
http://www.dia.unisa.it/dottorandi/decaro/
[3] Instituto de Computação, UNICAMP, Brazil
mochetti@ic.unicamp.br
http://www.ic.unicamp.br/~mochetti/

Abstract. The notion of inner-product encryption (IPE), introduced by Katz, Sahai, and Waters at Eurocrypt 2008, is a generalization of identity-based encryption in which ciphertexts and secret keys are associated to vectors in some finite field. In an IPE scheme, a ciphertext can only be decrypted by a secret key if the vector associated with the latter is orthogonal to that of the ciphertext. In its hierarchical version, first proposed by Okamoto and Takashima (Asiacrypt'09), there exists an additional delegation mechanism which allows users to delegate their decryption capabilities to other users in the system. In this paper, we propose the first construction of a hierarchical inner-product encryption (HIPE) scheme based on lattices assumptions. To achieve this goal, we extend the lattice-based IPE scheme by Agrawal, Freeman, and Vaikuntanathan (Asiacrypt'11) to the hierarchical setting by employing basis delegation technics by Peikert et al. (Eurocrypt'10) and by Agrawal et al. (Eurocrypt'10). As the underlying IPE scheme, our new scheme is shown to be weak selective secure based on the difficulty of the learning with errors (LWE) problem in the standard model, as long as the total number of levels in the hierarchy is a constant. As an application, we show how our new primitive can be used to build new chosen-ciphertext secure IPE and wildcarded identity-based encryption schemes.

Keywords. Lattice-based cryptography, inner product, functional cryptography, hierarchical.

1 Introduction

Functional encryption has become quite popular in the last few years because it provides the system administrator with a fine-grained control over the decryption capabilities of its users. Such ability makes functional encryption schemes appealing in many emerging applications such as cloud services where the notion of public-key encryption reveals its inadequacy.

Even though several different flavors of functional encryption have appeared in the literature (see [7,9,13]), it was only recently that a systematic study of this notion has

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 121–138, 2012.

been proposed by Boneh, Sahai and Waters [8]. In their notion, each functional encryption scheme is associated with a functionality, which defines the set of admissible functions of the plaintext to which a secret decryption key may be linked. More specifically, in a functional encryption system for functionality $F(\cdot,\cdot)$, an authority holding a master secret key msk, can generate a key d_k that enables the computation of the function $F(k,\cdot)$ on the encrypted data. Then, using d_k the decryptor can compute $F(k,x)$ from an encryption of x. Important examples of functional encryption are attribute-based encryption (ABE) [20,12] and predicate encryption (PE) [9,13].

In this paper, we focus on the notion of inner-product encryption (IPE), introduced by Katz, Sahai, and Waters at Eurocrypt 2008 [13], which is a generalization of identity-based encryption. In an IPE scheme, ciphertexts and secret keys are associated to vectors in some finite field and a ciphertext can only be decrypted by a secret key if the vector associated with the latter is orthogonal to that of the ciphertext. In its hierarchical version (HIPE), first proposed by Okamoto and Takashima [16], there exists an additional delegation mechanism which allows users to delegate their decryption capabilities to other users in the system. As pointed out in [13], IPE is a very powerful primitive as it can be used to support conjunction, subset and range queries on encrypted data as well as disjunctions, polynomial evaluation, and CNF and DNF formulas. Moreover, the delegation mechanism proposed by Okamoto and Takashima [16] extends even further the capabilities of this primitive.

Since its introduction, there has been an extensive amount of work on the construction of IPE and HIPE schemes, most of them based on bilinear groups. The first IPE scheme was proposed by Katz et al. [13]. They proved the security of their scheme in the selective model, where the adversary must commit to the input on which it wishes to be challenged before seeing the public parameters, under variants of the subgroup decisional assumption in composite order bilinear groups. Later on, Okamoto et al. [16] were able to move to prime order bilinear groups introducing at the same time the concept of HIPE primitive. There the security was still proved in the selective model. More recently, Okamoto et al. [17] presented a HIPE scheme that achieves full security under the standard d-linear assumption on prime order bilinear groups.

In the lattice-based setting, the only known construction of an inner product scheme is due to Agrawal et al. [3], which was shown to be weak selective secure based on the difficulty of the learning with errors (LWE) problem. Informally, in a weak selective secure IPE scheme, the set of secret keys to which the adversary can query is more restricted than the one allowed in a standard selective secure definition.

Our Contributions. In this paper, we propose the first construction of a hierarchical inner-product encryption scheme based on lattices assumptions. To achieve this goal, we extend the lattice-based IPE scheme by Agrawal et al. [3] to the hierarchical setting by employing basis delegation technics by Peikert et al. [10] and by Agrawal et al. [2]. As the underlying IPE scheme, our new scheme is shown to be weak selective secure based on the difficulty of the learning with errors (LWE) problem in the standard model, as long as the total number of levels in the hierarchy is a constant. As an application, we show how our new primitive can be used to build new chosen-ciphertext secure IPE and wildcarded identity-based encryption schemes based on lattices.

2 Definitions

In this section we introduce hierarchical inner-product encryption primitive and the tools that we will use to implement it. In doing so, we adopt the same notation and definition style used in [16,14,3].

Notation. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q and we represent $\mathbb{Z}_q$ as integers in $(q/2, q/2]$. We let $\mathbb{Z}_q^{n\times m}$ denote the set of $n \times m$ matrices with entries in $\mathbb{Z}_q$. We use bold capital letters (e.g. $\boldsymbol{A}$) to denote matrices, bold lowercase letters (e.g. $\boldsymbol{w}$) to denote vectors that are components of our encryption scheme. The notation $\boldsymbol{A}^\top$ denotes the transpose of the matrix $\boldsymbol{A}$. When we say a matrix defined over $\mathbb{Z}_q$ has *full rank*, we mean that it has full rank modulo each prime factor of q. If $\boldsymbol{A}_1$ is an $n \times m$ matrix and $\boldsymbol{A}_2$ is an $n \times m'$ matrix, then $[\boldsymbol{A}_1 \| \boldsymbol{A}_2]$ denotes the $n\times(m+m')$ matrix formed by concatenating $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$. If $\boldsymbol{w}_1$ is a length m vector and $\boldsymbol{w}_2$ is a length m' vector, then we let $[\boldsymbol{w}_1 \| \boldsymbol{w}_2]$ denote the length $(m+m')$ vector formed by concatenating $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. However, when doing matrix-vector multiplication we always view vectors as column vectors. We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we use $\mathrm{negl}(n)$ to denote a negligible function of n. We say $f(n)$ is *polynomial* if it is $O(n^c)$ for some $c > 0$, and we use $\mathrm{poly}(n)$ to denote a polynomial function of n. We say an event occurs with *overwhelming probability* if its probability is $1 - \mathrm{negl}(n)$. The function $\lg x$ is the base 2 logarithm of x. The notation $\lfloor x \rceil$ denotes the nearest integer to x, rounding towards 0 for half-integers. The norm of a matrix $\boldsymbol{R} \in \mathbb{R}^{k\times m}$ is defined as $\|\boldsymbol{R}\| := \sup_{\|\boldsymbol{u}\|=1} \|\boldsymbol{R}\boldsymbol{u}\|$. The notation $[d]$ denotes the set of positive integers $\{1, 2, \ldots, d\}$.

2.1 Hierarchical Inner-Product Encryption

Let $\boldsymbol{\mu}$ be a tuple of positive integers $\boldsymbol{\mu} = (\ell, d; \mu_1, \ldots, \mu_d)$ such that $\sum_{i\in[d]} \mu_i = \ell$. We call $\boldsymbol{\mu}$ an *hierarchical format* of depth d. Let $\Sigma_{|h} = (\Sigma_1 \times \ldots \times \Sigma_h)$ where $h \leq d$ and $\Sigma_i = \mathbb{F}_N^{\mu_i}$ for finite field $\mathbb{F}_N$ of order N. A *hierarchical predicate* $f_{\boldsymbol{v}}$, with $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_h) \in \Sigma_{|h}$, is defined as follows: $f_{\boldsymbol{v}}(\boldsymbol{w}) = 1$, with $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t) \in \Sigma_{|t}$, if and only if $h \leq t$ and for all $i \in [h]$ we have that $\langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle = 0$. An *hierarchical inner-product encryption* with hierarchical format $\boldsymbol{\mu}$ is defined by the following algorithms:

$\mathsf{Setup}(1^\lambda, \boldsymbol{\mu})$. Takes as input security parameter λ and hierarchical format $\boldsymbol{\mu}$ and outputs public parameters mpk and master secret key msk.

$\mathsf{Derive}(\boldsymbol{mpk}, d_{\boldsymbol{v}}, \boldsymbol{v}_t)$. Takes as input the master public key mpk, the secret key for the vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1}) \in \Sigma_{|t-1}$, and a vector $\boldsymbol{v}_t \in \Sigma_t$, and outputs a secret key $d_{\boldsymbol{v}'}$ for the vector $\boldsymbol{v}' = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1}, \boldsymbol{v}_t)$.

$\mathsf{Enc}(\boldsymbol{mpk}, \mathfrak{m}, \boldsymbol{w} = (\boldsymbol{w_1}, \ldots, \boldsymbol{w_t}) \in \boldsymbol{\Sigma_{|t}})$. Takes as input public parameters $\mathfrak{m}$ in some associated message space, public parameters mpk and an attribute vector $\boldsymbol{w}$ and outputs a ciphertext C.

$\mathsf{Dec}(\boldsymbol{mpk}, \boldsymbol{C}, d_{\boldsymbol{v}})$. Takes as input public parameters mpk, ciphertext C and secret key $d_{\boldsymbol{v}}$ and outputs the message $\mathfrak{m}$. We make the following consistency requirement. Suppose ciphertext C is obtained by running Enc on input mpk, message $\mathfrak{m}$ and attribute vector $\boldsymbol{w}$ and that $d_{\boldsymbol{v}}$ is a secret key for attribute vector $\boldsymbol{v}$ obtained through a

sequence of Derive calls using the same mpk. Then Dec, on input mpk, C and $d_{\boldsymbol{v}}$, returns $\mathfrak{m}$, except with negligible probability, if and only if $f_{\boldsymbol{v}}(\boldsymbol{w}) = 1$.

Security Definition. Security is modeled by means of a game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$. In this work, we achieve *selective attribute* security, meaning that $\mathcal{A}$ must declare its *challenge attribute vectors* $(\boldsymbol{w}_0, \boldsymbol{w}_1)$ before seeing the public key. Moreover, $\mathcal{A}$ is allowed to ask queries before and even after seeing the challenge ciphertext but it is required $\mathcal{A}$ to ask for keys of predicates $\boldsymbol{v}$ that cannot decrypt the challenge ciphertext; that is, for which $f_{\boldsymbol{v}}(\boldsymbol{w}_0) = f_{\boldsymbol{v}}(\boldsymbol{w}_1) = 0$. This notion is called *weak attribute hiding*. Specifically, the game is defined in the following way:

Init. $\mathcal{A}$ is given hierarchical format $\boldsymbol{\mu}$ of depth d and outputs challenge vectors $\boldsymbol{w}_0$, $\boldsymbol{w}_1 \in \Sigma_{|h}$.

Setup. The challenger $\mathcal{C}$ runs the Setup algorithm to generate public parameters mpk which it gives to the adversary $\mathcal{A}$.

Phase 1. $\mathcal{A}$ is given oracle access to Derive$(mpk, msk, \cdot)$. Then, $\mathcal{A}$ can delegate secret keys directly by invoking the Derive algorithm.

Challenge. $\mathcal{A}$ gives a pair of message $(\mathfrak{m}_0, \mathfrak{m}_1)$ to $\mathcal{C}$. We require that for all attribute vectors $\boldsymbol{v}$ derivable from any revealed secret key in Phase 1, $f_{\boldsymbol{v}}(\boldsymbol{w}_0) = f_{\boldsymbol{v}}(\boldsymbol{w}_1) = 0$. Then $\mathcal{C}$ chooses random $\eta \xleftarrow{\$} \{0,1\}$, encrypts $\mathfrak{m}_\eta$ under $\boldsymbol{w}_\eta$ and sends the resulting ciphertext to $\mathcal{A}$.

Phase 2. The same as Phase 1 with the same restriction of the Challenge phase.

Guess. $\mathcal{A}$ must output a guess β' for β. The advantage of $\mathcal{A}$ is defined to be $\Pr[\eta' = \eta] - \frac{1}{2}$.

Definition 1. An hierarchical inner-product encryption scheme is *weak attribute hiding-selective attribute* secure (IND-wAH-sAT-HIPE-CPA for short), if all polynomial time adversaries achieve at most a negligible (in λ) advantage in the previous security game. ∎

2.2 Lattices

In this section we collect results from [2,3,5,15,11,10] that we will need for our construction and the proof of security.

An m-dimensional lattice Λ is a full-rank discrete subgroup of $\mathbb{R}^m$. A basis of Λ is a linearly independent set of vectors whose span is Λ. We will focus on integer lattices and among these we will focus on the q-ary lattices defined as follows: for any integer $q \geq 2$ and any $\boldsymbol{A} \in \mathbb{Z}_q^{n \times m}$, we define

$$\Lambda_q^{\perp}(\boldsymbol{A}) := \{\boldsymbol{e} \in \mathbb{Z}_m : \boldsymbol{A} \cdot \boldsymbol{e} = 0 \mod q\}$$

$$\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}) := \{\boldsymbol{e} \in \mathbb{Z}_m : \boldsymbol{A} \cdot \boldsymbol{e} = \boldsymbol{u} \mod q\}$$

$$\Lambda_q(\boldsymbol{A}) := \{\boldsymbol{e} \in \mathbb{Z}_m : \exists\, \boldsymbol{s} \in \mathbb{Z}_q^m \text{ with } \boldsymbol{A}^t \cdot \boldsymbol{s} = \boldsymbol{e} \mod q\}.$$

The lattice $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A})$ is a coset of $\Lambda_q^{\perp}(\boldsymbol{A})$; namely, $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}) = \Lambda_q^{\perp}(\boldsymbol{A}) + \boldsymbol{t}$ for any $\boldsymbol{t}$ such that $\boldsymbol{A} \cdot \boldsymbol{t} = \boldsymbol{u} \mod q$.

Gram-Schmidt Norm. Let $\boldsymbol{S} = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$ be a set of vectors in $\mathbb{R}^m$. Let $\|\boldsymbol{S}\|$ denotes the length of the longest vector in $\boldsymbol{S}$, i.e., $\max_{1 \le i \le k} \|\boldsymbol{s}_i\|$, and $\widetilde{\boldsymbol{S}} := \widetilde{\boldsymbol{s}}_1, \ldots, \widetilde{\boldsymbol{s}}_k \subset \mathbb{R}^m$ denotes the *Gram-Schmidt orthogonalization* of the vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k$. We refer to $\|\widetilde{\boldsymbol{S}}\|$ as the *Gram-Schmidt norm* of $\boldsymbol{S}$.

Gaussian Distributions. Let L be a discrete subset of $\mathbb{Z}^n$. For any vector $\boldsymbol{c} \in \mathbb{R}^n$ and any positive parameter $\sigma \in \mathbb{R}_{>0}$, let $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{w}) := \exp\left(-\pi\|x - \boldsymbol{c}\|^2/\sigma^2\right)$ be the Gaussian function on $\mathbb{R}^n$ with center $\boldsymbol{c}$ and parameter σ. Let $\rho_{\sigma,\boldsymbol{c}}(L) := \sum_{\boldsymbol{w} \in L} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{w})$ be the discrete integral of $\rho_{\sigma,\boldsymbol{c}}$ over L, and let $\mathcal{D}_{L,\sigma,\boldsymbol{c}}$ be the discrete Gaussian distribution over L with center $\boldsymbol{c}$ and parameter σ. Specifically, for all $\boldsymbol{v} \in L$, we have $\mathcal{D}_{L,\sigma,\boldsymbol{c}}(\boldsymbol{v}) = \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{v})}{\rho_{\sigma,\boldsymbol{c}}(L)}$. For notational convenience, $\rho_{\sigma,0}$ and $\mathcal{D}_{L,\sigma,0}$ are abbreviated as ρ_σ and $\mathcal{D}_{L,\sigma}$ respectively. The following lemma captures standard properties of these distributions.

Lemma 2. *Let $q \ge 2$ and let $\boldsymbol{A}$ be a matrix in $\mathbb{Z}_q^{n \times m}$ with $m > n$. Let $\boldsymbol{T}_{\boldsymbol{A}}$ be a basis for $\Lambda_q^\perp(\boldsymbol{A})$ and $\sigma \ge \|\widetilde{\boldsymbol{T}_{\boldsymbol{A}}}\| \cdot \omega(\sqrt{\log m})$. Then for $\boldsymbol{c} \in \mathbb{R}^m$ and $\boldsymbol{u} \in \mathbb{Z}_q^n$:*

1. $\Pr\left[\|\boldsymbol{w} - \boldsymbol{c}\| > \sigma\sqrt{m} \ : \ \boldsymbol{w} \xleftarrow{\$} \mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}\right] \le \mathrm{negl}(n)$
2. *A set of $O(m \log m)$ samples from $\mathcal{D}_{\Lambda_q^\perp(A),\sigma}$ contains a full rank set in $\mathbb{Z}^m$, except with negligible probability.*
3. *There is a PPT algorithm* $\mathsf{SampleGaussian}(\boldsymbol{A}, \boldsymbol{T}_{\boldsymbol{A}}, \sigma, \boldsymbol{c})$ *that returns $\boldsymbol{x} \in \Lambda_q^\perp(\boldsymbol{A})$ drawn from a distribution statistically close to $\mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}$.*
4. *There is a PPT algorithm* $\mathsf{SamplePre}(\boldsymbol{A}, \boldsymbol{T}_{\boldsymbol{A}}, \boldsymbol{u}, \sigma)$ *that returns $\boldsymbol{x} \in \Lambda_q^\perp(A)$ sampled from a distribution statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}),\sigma}$, whenever $\Lambda_q^{\boldsymbol{u}}(A)$ is not empty.*

∎

The Norm of a Random Matrix. The following lemmata can be used to bound the norm of a random matrix in $\{-1, 1\}^{m \times m}$.

Lemma 3. *([2, Lemma 15]) Let $\boldsymbol{R}$ be a $k \times m$ matrix chosen at random from $\{-1, 1\}^{k \times m}$. Then* $\Pr\left[\|\boldsymbol{R}\| > 12\sqrt{k+m}\right] < e^{-(k+m)}$. ∎

Lemma 4. *([2, Lemma 16]) Let $\boldsymbol{u} \in \mathbb{R}^m$ be some vector of norm 1. Let $\boldsymbol{R}$ be a $k \times m$ matrix chosen at random from $\{-1, 1\}^{k \times m}$. Then* $\Pr[\|\boldsymbol{R}\boldsymbol{u}\| > \sqrt{k}\omega(\sqrt{\log k})] < \mathrm{negl}(k)$. ∎

Sampling Algorithms. Following [2,10,4,5] we will need the following algorithms to sample short vectors and random basis from specific lattices.

Algorithm ToBasis. Micciancio and Goldwassser [31] showed that a full-rank set $\boldsymbol{S}$ in a lattice Λ can be converted into a basis T for Λ with an equally low Gram-Schmidt norm.

Lemma 5. *([31, Lemma 7.1]) Let Λ be an m-dimensional lattice. There is a deterministic polynomial-time algorithm that, given an arbitrary basis of Λ and a full-rank set $\boldsymbol{S} = \boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$ in Λ, returns a basis T of Λ satisfying $\|\widetilde{\boldsymbol{T}}\| \le \|\widetilde{\boldsymbol{S}}\|$ and $\|\boldsymbol{T}\| \le \|\boldsymbol{S}\|\sqrt{m}/2$* ∎

Algorithm TrapGen. Ajtai [4] and later Alwen and Peikert [5] showed how to sample an essentially uniform matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$ along with a basis $\boldsymbol{S}$ of $\Lambda_q^{\perp}(\boldsymbol{A})$ with low Gram-Schmidt norm.

Theorem 6. *([5, Theorem 3.2] with $\delta = 1/3$) Let q, n, m be positive integers with $q \geq 2$ and $m \geq 6n\lg q$. There is a probabilistic polynomial-time algorithm* $\mathsf{TrapGen}(q, n, m)$ *that outputs a pair $(\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}, \boldsymbol{S} \in \mathbb{Z}^{m\times m})$ such that $\boldsymbol{A}$ is statistically close to uniform in $\mathbb{Z}_q^{n\times m}$ and $\boldsymbol{S}$ is a basis for $\Lambda_q^{\perp}(\boldsymbol{A})$, satisfying $\|\widetilde{\boldsymbol{S}}\| \leq O(\sqrt{n\log q})$ and $\|\boldsymbol{S}\| \leq O(n\log q)$ w.o.p. in n.* ∎

We let $\sigma_{\mathrm{TG}} = O(\sqrt{n\log q})$ denote the maximum with high probability Gram-Schmidt norm of a basis produced by $\mathsf{TrapGen}$.

Algorithm ExtendBasis. Peikert et al. [10] shows how to construct a basis for $\Lambda_q^{\perp}(\boldsymbol{A}\|\boldsymbol{B}\|\boldsymbol{C})$ from a basis for $\Lambda_q^{\perp}(\boldsymbol{B})$.

Theorem 7. *For $i = 1, 2, 3$ let $\boldsymbol{A}_i$ be a matrix in $\mathbb{Z}_q^{n\times m_i}$ and let $\boldsymbol{A} := (\boldsymbol{A}_1\|\boldsymbol{A}_2\|\boldsymbol{A}_3)$. Let $\boldsymbol{T}_2$ be a basis of $\Lambda_q^{\perp}(\boldsymbol{A}_2)$. There is deterministic polynomial time algorithm* $\mathsf{ExtendBasis}(\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3, \boldsymbol{T}_2)$ *that outputs a basis T for $\Lambda_q^{\perp}(\boldsymbol{A})$ such that $\|\widetilde{\boldsymbol{T}}\| = \|\widetilde{\boldsymbol{T}_2}\|$* ∎

Algorithm SampleLeft. The algorithm takes as input a full rank matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$, a short basis $\boldsymbol{T_A}$ of $\Lambda_q^{\perp}(\boldsymbol{A})$, a matrix $\boldsymbol{B} \in \mathbb{Z}_q^{n\times m_1}$, a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter σ. Let $\boldsymbol{F} := (\boldsymbol{A}\|\boldsymbol{B})$, then the algorithm outputs a vector $\boldsymbol{e} \in \mathbb{Z}^{m+m_1}$ in the coset $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{F})$.

Theorem 8. *([2, Theorem 17], [10, Lemma 3.2]) Let $q > 2, m > n$ and $\sigma > \|\boldsymbol{T_A}\| \cdot \omega(\sqrt{\log(m+m_1)})$. Then* $\mathsf{SampleLeft}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{T_A}, \boldsymbol{u}, \sigma)$ *outputs a vector $\boldsymbol{e} \in \mathbb{Z}^{m+m_1}$ statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\boldsymbol{F}),\sigma}$.* ∎

Algorithm SampleRight. The algorithm takes as input matrices $\boldsymbol{A} \in \mathbb{Z}_q^{n\times k}$ and $\boldsymbol{R} \in \mathbb{Z}^{k\times m}$, a full rank matrix $\boldsymbol{B} \in \mathbb{Z}_q^{n\times m}$ and a short basis $\boldsymbol{T_B}$ of $\Lambda_q^{\perp}(\boldsymbol{B})$, a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter σ. Let $\boldsymbol{F} := (\boldsymbol{A}\|\boldsymbol{AR} + \boldsymbol{B})$, then the algorithm outputs a vector $\boldsymbol{e} \in \mathbb{Z}^{k+m}$ in the coset $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{F})$.

Often the matrix $\boldsymbol{R}$ given to the algorithm as input will be a random matrix in $\{-1, 1\}^{m\times m}$. Let S^m be the m-sphere $\{\boldsymbol{x} \in \mathbb{R}^{m+1} : \|\boldsymbol{x}\| = 1\}$. We define $s_{\boldsymbol{R}} := \|\boldsymbol{R}\| = \sup_{\boldsymbol{x}\in S^{m-1}} \|\boldsymbol{R}\cdot\boldsymbol{x}\|$.

Theorem 9. *([2, Theorem 19]) Let $q > 2, m > n$ and $\sigma > \|\boldsymbol{T_B}\|\cdot s_R \cdot \omega(\sqrt{\log(k+m)})$. Then* $\mathsf{SampleRight}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{R}, \boldsymbol{T_B}, \boldsymbol{u}, \sigma)$ *outputs a vector $\boldsymbol{e} \in Z^{k+m}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\boldsymbol{F}),\sigma}$.* ∎

The LWE Problem. The Learning with Errors problem, or LWE, is the problem of determining a secret vector over $\mathbb{F}_q$ given a polynomial number of noisy inner products. The decision variant is to distinguish such samples from random. More formally, we define the (average-case) problem as follows:

Definition 10. ([19]) Let $n \geq 1$ and $q \geq 2$ be integers, and let χ be a probability distribution on $\mathbb{Z}_q$. For $\boldsymbol{r} \in \mathbb{Z}_q^n$, let $A_{\boldsymbol{r},\chi}$ be the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing a vector $\boldsymbol{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in \mathbb{Z}_q$ according to χ, and outputting $(\boldsymbol{a}, \langle\boldsymbol{a}, \boldsymbol{r}\rangle + e)$.

(a) The *search*-$\mathsf{LWE}_{q,n,\chi}$ problem is: for uniformly random $\boldsymbol{r} \in \mathbb{Z}_q^n$, given a $\mathrm{poly}(n)$ number of samples from $A_{\boldsymbol{r},\chi}$, output $\boldsymbol{r}$.
(b) The *decision*-$\mathsf{LWE}_{q,n,\chi}$ problem is: for uniformly random $\boldsymbol{r} \in \mathbb{Z}_q^n$, given a $\mathrm{poly}(n)$ number of samples that are either (all) from $A_{\boldsymbol{r},\chi}$ or (all) uniformly random in $\mathbb{Z}_q^n \times Z_q$, output 0 if the former holds and 1 if the latter holds.

We say the *decision*-$\mathsf{LWE}_{q,n,\chi}$ problem is infeasible if for all polynomial-time algorithms $\mathcal{A}$, the probability that $\mathcal{A}$ solves the *decision*-LWE problem (over $\boldsymbol{r}$ and $\mathcal{A}$s random coins) is negligibly close to $1/2$ as a function of n. ∎

The hardness of the LWE problem is summarized in the following:

Definition 11. For $\alpha \in (0,1)$ and an integer $q > 2$, let $\overline{\Psi}_\alpha$ denote the probability distribution over $\mathbb{Z}_q$ obtained by choosing $x \in \mathbb{R}$ according to the normal distribution with mean 0 and standard deviation $\alpha/\sqrt{2\pi}$ and outputting $\lfloor qx \rceil$. ∎

Theorem 12. *([19]) Let n, q be integers and $\alpha \in (0,1)$ such that $q = \mathrm{poly}(n)$ and $\alpha q > 2\sqrt{n}$. If there exists an efficient (possibly quantum) algorithm that solves decision-$\mathsf{LWE}_{q,n,\overline{\Psi}_\alpha}$, then there exists an efficient quantum algorithm that approximates SIVP and GapSVP to within $\widetilde{O}(n/\alpha)$ in the worst case.* ∎

Theorem 13. *([18]) Let n, q be integers and $\alpha \in (0,1)$, and $q = \sum_i q_i \leq 2^{n/2}$, where the q_i are distinct primes satisfying $\omega(\log n)/\alpha \leq q_i \leq \mathrm{poly}(n)$. If there exists an efficient (classical) algorithm that solves decision-$\mathsf{LWE}_{q,n,\overline{\Psi}_\alpha}$, then there exists an efficient (classical) algorithm that approximates GapSVP to within $\widetilde{O}(n/\alpha)$ in the worst case.* ∎

The following lemma will be used to show correctness of decryption.

Lemma 14. *([2, Lemma 12]) Let $\mathbf{e}$ be some vector in $\mathbb{Z}^m$ and let $\boldsymbol{v} \leftarrow \overline{\Psi}_\alpha^m$. Then the quantity $|\langle \mathbf{e}, \boldsymbol{v} \rangle|$ when treated as an integer in $(-q/2, q/2]$ satisfies $|\langle \mathbf{e}, \boldsymbol{v} \rangle| \leq \|\mathbf{e}\| \cdot \left(q\alpha \cdot \omega(\sqrt{\log m}) + \sqrt{m}/2\right)$ w.o.p. (in m).* ∎

3 Hierarchical Inner Product Encryption Scheme

This section describes our hierarchical inner-product encryption scheme based on [3].

Intuitions. For hierarchical format $\boldsymbol{\mu} = (\ell, d; \mu_1, \ldots, \mu_d)$, the public parameters will contain random matrices $(\boldsymbol{A}, \{\boldsymbol{A}_{i,j,\gamma}\})$ in $\mathbb{Z}_q^{n \times m}$. The master secret key is a trapdoor $\boldsymbol{T}_{\boldsymbol{A}}$ for $\boldsymbol{A}$. To generate a secret key for vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$ at depth $t \leq d$ we use the matrix:

$$F_{\boldsymbol{v}} = \left(\boldsymbol{A} \| \sum_{j \in [\mu_1]} \sum_{\gamma=0}^{k} v_{1,j,\gamma} \cdot \boldsymbol{A}_{1,j,\gamma} \| \ldots \| \sum_{j \in [\mu_t]} \sum_{\gamma=0}^{k} v_{t,j,\gamma} \cdot \boldsymbol{A}_{t,j,\gamma} \right) \in \mathbb{Z}_q^{n \times (t+1)m} \tag{1}$$

where each $v_{i,j}$ is r-decomposed for a certain fixed r and $k = \lfloor \log_r q \rfloor$. Then the secret key for $\boldsymbol{v}$ is a short basis for the lattice $\Lambda_q^\perp(F_{\boldsymbol{v}})$. By using the short basis for $\Lambda_q^\perp(F_{\boldsymbol{v}})$ is

possible to generate a random short basis for $\Lambda_q^{\perp}(F_{\boldsymbol{v}\|\boldsymbol{v}_{t+1}})$. This provides the delegation mechanism.

Let us sketch briefly the security reduction. For challenge vector $\boldsymbol{w}^{\star} = (\boldsymbol{w}_1^{\star}, \ldots, \boldsymbol{w}_{t^{\star}}^{\star})$, the simulator chooses the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ uniformly at random in $\mathbb{Z}_q^{n\times m}$ and construct the matrices $\boldsymbol{A}_{i,j,\gamma}$ as follows:

$$\boldsymbol{A}_{i,j,\gamma} = \boldsymbol{A}\boldsymbol{R}_{i,j,\gamma} - r^{\gamma} w_{i,j}^{\star} \boldsymbol{B}$$

where $\boldsymbol{R}_{i,j,\gamma} \in \{-1,1\}^{m\times m}$. Since the matrices $\boldsymbol{A}, \{\boldsymbol{R}_{i,j,\gamma}\}$ are uniform and independent in $\mathbb{Z}_q^{n\times m}$, we have that the $\boldsymbol{A}_{i,j,\gamma}$'s are uniform in $\mathbb{Z}_q^{n\times m}$ as in the real system. Moreover the simulator has a trapdoor $\boldsymbol{T}_{\boldsymbol{B}}$ for $\Lambda_q^{\perp}(\boldsymbol{B})$ but no trapdoor for $\Lambda_q^{\perp}(\boldsymbol{A})$. To generate a secret key for vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$, the simulator must produce a short basis for $\Lambda_q^{\perp}(F_{\boldsymbol{v}})$ where

$$F_{\boldsymbol{v}} = \left(\boldsymbol{A}\|\boldsymbol{A}\overline{\boldsymbol{R}_1} - \langle \boldsymbol{v}_1, \boldsymbol{w}_1^{\star}\rangle \boldsymbol{B}\| \ldots \|\boldsymbol{A}\overline{\boldsymbol{R}_t} - \langle \boldsymbol{v}_t, \boldsymbol{w}_t^{\star}\rangle \boldsymbol{B}\right)$$

Then let

$$\begin{aligned}
\overline{\boldsymbol{R}_i} &= \sum_{j\in[\mu_i]} \sum_{\gamma=0}^{k} v_{i,j,\gamma} \cdot \boldsymbol{R}_{i,j,\gamma} \quad \in \mathbb{Z}_q^{m\times m} \\
\overline{\boldsymbol{R}} &= \left[\overline{\boldsymbol{R}_1}\| \ldots \|\overline{\boldsymbol{R}_t}\right] \quad \in \mathbb{Z}_q^{m\times t\cdot m} \\
\boldsymbol{B}_{\boldsymbol{v}} &= [-\langle \boldsymbol{v}_1, \boldsymbol{w}_1^{\star}\rangle \boldsymbol{B}\| \ldots \| - \langle \boldsymbol{v}_t, \boldsymbol{w}_t^{\star}\rangle \boldsymbol{B}] \quad \in \mathbb{Z}_q^{n\times t\cdot m}
\end{aligned}$$

Thus $F_{\boldsymbol{v}}$ can be written as:

$$F_{\boldsymbol{v}} = \left(\boldsymbol{A}\|\boldsymbol{A}\overline{\boldsymbol{R}} + \boldsymbol{B}_{\boldsymbol{v}}\right) \quad \in \mathbb{Z}_q^{n\times(t+1)m} . \tag{2}$$

When $\boldsymbol{v}$ is not a prefix of $\boldsymbol{w}$ meaning that there exists an index i such that $\langle \boldsymbol{v}_i, \boldsymbol{w}_i^{\star}\rangle \neq 0$, the simulator can then extend $\boldsymbol{T}_{\boldsymbol{B}}$ to a short basis for the entire lattice $\Lambda_q^{\perp}(\boldsymbol{B}_{\boldsymbol{v}})$. The simulator can now generate short vectors in $\Lambda_q^{\perp}(F_{\boldsymbol{v}})$ using algorithm SampleRight, which is sufficient for constructing a short basis for $\Lambda_q^{\perp}(F_{\boldsymbol{v}})$, as required. When $\boldsymbol{v}$ is a prefix of $\boldsymbol{w}$, meaning that for each $i = 1, \ldots, t$, $\langle \boldsymbol{v}_i, \boldsymbol{w}_i^{\star}\rangle = 0$, then the matrix $F_{\boldsymbol{v}}$ no longer depends on $\boldsymbol{B}$ and the simulators trapdoor disappears. Consequently, the simulator can generate secret keys for all vectors other than prefixes of $\boldsymbol{w}^{\star}$. As we will see, for $\boldsymbol{w}^{\star}$ the simulator can produce a challenge ciphertext that helps it solve the given LWE challenge.

3.1 Sampling a Random Basis

In this Section we describe the algorithms that we will use to realize the delegation mechanism and for the simulation. Following [2,10], let Λ be an m-dimensional lattice and let $\mathcal{O}(\Lambda, \sigma)$ be an algorithm that generates independent samples from a distribution statistically close to $\mathcal{D}_{\Lambda,\sigma}$. The following algorithm called $\mathsf{SampleBasis}^{\mathcal{O}}(\Lambda, \sigma)$ uses $\mathcal{O}$ to generate a basis T of Λ in the following way:

1. For $i = 1, \ldots, m$, generate $v \xleftarrow{\$} \mathcal{O}(\Lambda, \sigma)$, if v is independent of $\{v_1, \ldots, v_{i-1}\}$, set $v_i \leftarrow v$, if not, repeat.
2. Convert the set of independent vectors $v_1, \ldots, v_m$ to a basis T using Lemma 6 (and using some canonical basis of Λ) and output T.

The following theorem summarizes properties of this algorithm.

Lemma 15. *For $\sigma > \tilde{bl}(\Lambda)\omega(\sqrt{\log m})$ algorithm $\mathsf{SampleBasis}^{\mathcal{O}}(\Lambda, \sigma)$ satisfies the following properties:*

1. *Step 1 requires at most $O(m \log m)$ w.h.p and $2m$ samples in expectation.*
2. *With overwhelming probability $\|\widetilde{T}\| \leq \|T\| \leq \sigma\sqrt{m}$.*
3. *Up to a statistical distance, the distribution of T does not depend on the implementation of $\mathcal{O}$. That is, the random variable $\mathsf{SampleBasis}^{\mathcal{O}}(\Lambda, \sigma)$ is statistically close to $\mathsf{SampleBasis}^{\mathcal{O}'}(\Lambda, \sigma)$ for any algorithm $\mathcal{O}'$ that samples from a distribution statistically close to $\mathcal{D}_{\Lambda,\sigma}$.*

∎

Algorithm SampleBasisLeft. We are interested in the lattice $\Lambda_q^{\perp}(F_{\boldsymbol{v}})$ where $F_{\boldsymbol{v}}$ is defined in (1) for $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$. Write $F_{\boldsymbol{v}} = (\boldsymbol{A}|\boldsymbol{M})$ for some matrices $\boldsymbol{A}$ and $\boldsymbol{M}$, then given a short basis $\boldsymbol{T_A}$ for $\Lambda_q^{\perp}(\boldsymbol{A})$ we can implement algorithm $\mathcal{O}(F_{\boldsymbol{v}}, \sigma)$ by invoking $\mathsf{SampleLeft}(\boldsymbol{A}, \boldsymbol{M}, \boldsymbol{T_A}, 0, \sigma)$. When $\sigma > \|\widetilde{\boldsymbol{T_A}}\| \cdot \omega(\sqrt{\log((t+1)m)})$, Theorem 8 shows that the resulting vector is distributed statistically close to $\mathcal{D}_{\Lambda_q^{\perp}(F_{\boldsymbol{v}}),\sigma}$ as required for SampleBasis. Using the above algorithm in algorithm SampleBasis leads to an algorithm to sample a random basis of $\Lambda_q^{\perp}(F_{\boldsymbol{v}})$ given a short basis of $\boldsymbol{A}$. We refer to this algorithm as SampleBasisLeft and summarize its properties in the following corollary.

Corollary 16. Algorithm $\mathsf{SampleBasisLeft}(\boldsymbol{A}, \boldsymbol{M}, \boldsymbol{T_A}, \sigma)$ outputs a basis of $\Lambda_q^{\perp}(F_{\boldsymbol{v}})$ satisfying the three properties in Lemma 15 provided that $\boldsymbol{A}$ is rank n and $\sigma > \|\widetilde{\boldsymbol{T_A}}\| \cdot \omega\left(\sqrt{\log((t+1)m)}\right)$. ∎

Algorithm SampleBasisRight. In the simulation, the matrix $F_{\boldsymbol{v}}$ is defined as in (2). In this case, given a short basis $\boldsymbol{T_B}$ for $\Lambda_q^{\perp}(\boldsymbol{B})$ we can implement algorithm $\mathcal{O}(F_{\boldsymbol{v}}, \sigma)$ as follows:

1. Using Theorem 7, extend basis $\boldsymbol{T_B}$ for $\Lambda_q^{\perp}(\boldsymbol{B})$ to a basis $\boldsymbol{T_{B_v}}$ for $\Lambda_q^{\perp}(\boldsymbol{B_v})$ such $\|\widetilde{\boldsymbol{T_{B_v}}}\| = \|\widetilde{\boldsymbol{T_B}}\|$.
2. Then run $\mathsf{SampleRight}(\boldsymbol{A}, \boldsymbol{B_v}, \overline{\boldsymbol{R}}, \boldsymbol{T_{B_v}}, 0, \sigma)$ and output the result. When $\boldsymbol{B_v}$ is rank n and $\boldsymbol{v}$ is not a prefix of $\boldsymbol{w}^{\star}$ the matrix $\boldsymbol{B_v}$ is rank n as required for SampleRight.

Let $s_{\overline{\boldsymbol{R}}} := \|\overline{\boldsymbol{R}}\|$ be the norm of the matrix $\overline{\boldsymbol{R}}$. When $\sigma > \|\widetilde{\boldsymbol{T_B}}\| \cdot s_{\overline{\boldsymbol{R}}} \cdot \omega(\sqrt{\log((t+1)m)})$, Theorem 9 shows that the resulting vector is distributed statistically close to $\mathcal{D}_{\Lambda_q^{\perp}(F_{\boldsymbol{v}}),\sigma}$ as required for SampleBasis. Using the above algorithm in algorithm SampleBasis leads to an algorithm to sample a random basis of $\Lambda_q^{\perp}(F_{\boldsymbol{v}})$ for $F_{\boldsymbol{v}}$ defined in (2) given a short basis of $\boldsymbol{B}$. We refer to this algorithm as SampleBasisRight and summarize its properties in the following corollary.

Corollary 17. Algorithm $\mathsf{SampleBasisRight}(\boldsymbol{A}, \boldsymbol{B_v}, \overline{\boldsymbol{R}}, \boldsymbol{T_B}, \sigma)$ outputs a basis of $\mathcal{D}_{\Lambda_q^\perp(F_{\boldsymbol{v}}),\sigma}$ satisfying the three properties in Lemma 15 provided that $\boldsymbol{B}$ is rank n, that $\boldsymbol{v}$ is not a prefix of $\boldsymbol{w}^\star$ and $\sigma > \|\widetilde{\boldsymbol{T_B}}\| \cdot s_{\overline{\boldsymbol{R}}} \cdot \omega(\sqrt{\log((t+1)m)})$. ∎

3.2 Our Construction

Let $n > 0$ be the security parameter and $\boldsymbol{\mu} = (\ell, d; \mu_1, \ldots, \mu_d)$ the hierarchical format. Let $q = q(n, \boldsymbol{\mu})$ and $m = m(n, \boldsymbol{\mu})$ be positive integers. Let $r = r(n, \boldsymbol{\mu}) \geq 2$ be and integer and define $k = k(n, \boldsymbol{\mu}) := \lfloor \log_r q \rfloor$. Our *hierarchical inner-product encryption* for hierarchical format $\boldsymbol{\mu}$ consists of the following algorithms.

$\mathsf{Setup}(1^n, \boldsymbol{\mu})$. On input a security parameter n, and an hierarchical format of depth d $\boldsymbol{\mu} = (\ell, d; \mu_1, \ldots, \mu_d)$, the algorithm generates public and secret parameters as follows: Use algorithm $\mathsf{TrapGen}(q, n, m)$ to select a uniformly random $n \times m$-matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times m}$ with a basis $\boldsymbol{T_A} \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^\perp(\boldsymbol{A})$ such that $\|\widetilde{\boldsymbol{T_A}}\| \leq O(\sqrt{n \log q})$. For $i \in [d]$, $j \in [\mu_i]$ and $\gamma = 0, \ldots, k$, choose uniformly random matrices $\boldsymbol{A}_{i,j,\gamma} \in \mathbb{Z}_q^{n \times m}$. Select a uniformly random vector $\boldsymbol{u} \in \mathbb{Z}_q^n$. Output $mpk = (\boldsymbol{A}, \{\boldsymbol{A}_{i,j,\gamma}\}, \boldsymbol{u})$ and $msk = \boldsymbol{T_A}$.

$\mathsf{Derive}(\boldsymbol{mpk}, \boldsymbol{d_v}, \boldsymbol{v_t})$. On input the master public key mpk, the secret key for the vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$, and the vector $\boldsymbol{v}_t$, the algorithm generates a secret key for the vector $\boldsymbol{v'} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$ as follows: Construct short basis for $\Lambda_q^\perp(F_{\boldsymbol{v'}})$ by invoking $\boldsymbol{S} \leftarrow \mathsf{SampleBasisLeft}(F_{\boldsymbol{v}}, \sum_{j \in [\mu_i]} \sum_{\gamma \in k} v_{t,j,\gamma} \cdot \boldsymbol{A}_{t,j,\gamma}, d_{\boldsymbol{v}}, \sigma_t)$ where $F_{\boldsymbol{v'}} = \left[F_{\boldsymbol{v}} \| \sum_{j \in [\mu_i]} \sum_{\gamma=0}^{k} v_{t,j,\gamma} \cdot \boldsymbol{A}_{t,j,\gamma}\right]$ and $d_{\boldsymbol{v}}$ is a short basis for $\Lambda_q^\perp(F_{\boldsymbol{v}})$. By Corollary 16, when $\sigma_t > \|\widetilde{d_{\boldsymbol{v}}}\| \cdot \omega(\sqrt{\log((t+1)m)})$ then $\|\widetilde{d_{\boldsymbol{v'}}}\| \leq \|d_{\boldsymbol{v'}}\| \leq \sigma_t \cdot \sqrt{(t+1)m}$. Output $d_{\boldsymbol{v'}} = \boldsymbol{S}$. Notice that, for the special case of the first level secret keys when $\boldsymbol{v}$ is the empty vector ϵ, we define $F_\epsilon := \boldsymbol{A}$ and $d_{\boldsymbol{v}} = msk$.

$\mathsf{Enc}(\boldsymbol{mpk}, \boldsymbol{w}, \mathfrak{m})$. On input the master public key mpk, the vector $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t)$, and the message $\mathfrak{m} \in \{0, 1\}$, the algorithm generates a ciphertext C as follows: Choose a uniformly random matrix $\boldsymbol{B} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ and $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$. Choose a noise vector $\boldsymbol{x} \leftarrow \overline{\Psi}_{\alpha_t}^m$ and a noise term $x \leftarrow \overline{\Psi}_{\alpha_t}$. Compute $\boldsymbol{c}_0 = \boldsymbol{A}^\top \boldsymbol{s} + \boldsymbol{x} \in \mathbb{Z}_q^m$. For $i \in [t]$, $j \in [\mu_i]$ and $\gamma = 0, \ldots, k$ choose a random matrix $\boldsymbol{R}_{i,j,\gamma} \in \{-1, 1\}^{m \times m}$ and compute $\boldsymbol{c}_{i,j,\gamma} = (\boldsymbol{A}_{i,j,\gamma} + r^\gamma w_{i,j} \boldsymbol{B})^\top \boldsymbol{s} + \boldsymbol{R}_{i,j,\gamma}^\top \boldsymbol{x} \in \mathbb{Z}_q^m$. Compute $c' = \boldsymbol{u}^\top \boldsymbol{s} + x + \mathfrak{m} \cdot \lfloor q/2 \rceil \in \mathbb{Z}_q$. Output $C = (\boldsymbol{c}_0, \{\boldsymbol{c}_{i,j,\gamma}\}, c')$.

$\mathsf{Dec}(\boldsymbol{mpk}, \boldsymbol{d_v}, \boldsymbol{C})$. On input the master public key mpk, the secret key for the vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$, and a ciphertext $C = (\boldsymbol{c}_0, \{\boldsymbol{c}_{i,j,\gamma}\}, c')$, the algorithm does the following: For $i \in [t]$ define the r-ary expansion of the vector $\boldsymbol{v}_i$ and compute $\boldsymbol{c}_{\boldsymbol{v}_i} = \sum_{j \in [\mu_i]} \sum_{\gamma=0}^{k} v_{i,j,\gamma} \cdot \boldsymbol{c}_{i,j,\gamma}$. Let $\boldsymbol{c} = [\boldsymbol{c}_0 \| \boldsymbol{c}_{\boldsymbol{v}_1} \| \ldots \| \boldsymbol{c}_{\boldsymbol{v}_t}]$. Set $\tau_t := \sigma_t \cdot \sqrt{(t+1)m} \cdot \omega(\sqrt{(t+1)m})$. Then $\tau_t \geq \|\widetilde{d_{\boldsymbol{v}}}\| \cdot \omega(\sqrt{(t+1)m})$. Compute $\boldsymbol{e_v} = \mathsf{SamplePre}(F_{\boldsymbol{v}}, d_{\boldsymbol{v}}, \boldsymbol{u}, \tau_t)$ and $z = c' - \boldsymbol{e_v}^\top \cdot \boldsymbol{c}$.

Interpreter z as in integer in $(-q/2, q/2]$, then output 0 if $|z| < q/4$ and 1 otherwise.

3.3 Correctness

Lemma 18. *For hierarchical format $\boldsymbol{\mu} = (\ell, d; \mu_1, \ldots, \mu_d)$ of depth d, suppose the parameters q and α_t, for each $t \in [d]$, are such that $q/\log q = \Omega(\sigma_t \cdot \mu \cdot \frac{r}{\log r} \cdot m^{3/2})$ and $\alpha_t \leq (t \cdot \log q \cdot \sigma_t \cdot \mu \cdot \frac{r}{\log r} \cdot m \cdot \omega(\sqrt{\log m}))^{-1}$, where $\mu = \max_{i \in [d]} \mu_i$. Moreover, for vectors $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$ and $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_h)$, let $d_{\boldsymbol{v}}$ be the basis of the lattice $F_{\boldsymbol{v}}$ obtained through a sequence of calls to the* Derive *algorithm, let $C \leftarrow \mathsf{Enc}(mpk, \boldsymbol{w}, \mathfrak{m})$ and $\mathfrak{m}' \leftarrow \mathsf{Dec}(mpk, d_{\boldsymbol{v}}, C)$.*

Then, if $f_{\boldsymbol{v}}(\boldsymbol{w}) = 1$, namely $\langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle = 0 \pmod q$ for each $i \in [t]$, then with overwhelming probability we have $\mathfrak{m}' = \mathfrak{m}$. ∎

Proof. Just for notation simplification, let

$$\widetilde{\boldsymbol{A}}_i = \left(\sum_{j \in [\mu]} \sum_{\gamma=0}^{k} v_{i,j,\gamma} \boldsymbol{A}_{i,j,\gamma} \right) \quad \text{and} \quad \widetilde{\boldsymbol{R}}_i = \sum_{j \in [\mu]} \sum_{\gamma=0}^{k} v_{i,j,\gamma} \boldsymbol{R}_{i,j,\gamma}^{\top} \boldsymbol{x} \, .$$

Then, for each $i \in [t]$, the decryption algorithm computes:

$$\begin{aligned} \boldsymbol{c}_{\boldsymbol{v}_i} &= \sum_{j \in [\mu]} \sum_{\gamma=0}^{k} v_{i,j,\gamma} \cdot \boldsymbol{c}_{i,j,\gamma} \\ &= \sum_{j \in [\mu]} \sum_{\gamma=0}^{k} v_{i,j,\gamma} \cdot \left[(\boldsymbol{A}_{i,j,\gamma} + r^{\gamma} w_{i,j} \boldsymbol{B})^{\top} \boldsymbol{s} + \boldsymbol{R}_{i,j,\gamma}^{\top} \boldsymbol{x} \right] \\ &= \widetilde{\boldsymbol{A}}_i^{\top} \boldsymbol{s} + \underbrace{\left(\sum_{j \in [\mu]} \sum_{\gamma=0}^{k} r^{\gamma} v_{i,j,\gamma} w_{i,j} \right)}_{\langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle} \boldsymbol{B}^{\top} \boldsymbol{s} + \widetilde{\boldsymbol{R}}_i \, . \end{aligned}$$

If $\langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle = 0$ then we have:

$$\boldsymbol{c}_{\boldsymbol{v}_i} = \widetilde{\boldsymbol{A}}_i^{\top} \boldsymbol{s} + \widetilde{\boldsymbol{R}}_i \pmod q \, .$$

Thus, if for each $i \in [t]$, $\langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle = 0$ then $\boldsymbol{c}$ can be written as:

$$\begin{aligned} \boldsymbol{c} &= [\boldsymbol{c}_0 \| \boldsymbol{c}_{\boldsymbol{v}_1} \| \ldots \| \boldsymbol{c}_{\boldsymbol{v}_t}] \\ &= \left[\boldsymbol{A} \| \widetilde{\boldsymbol{A}}_1 \| \ldots \| \widetilde{\boldsymbol{A}}_t \right]^{\top} \boldsymbol{s} + \left[\boldsymbol{x} \| \widetilde{\boldsymbol{R}}_1 \| \ldots \| \widetilde{\boldsymbol{R}}_t \right]^{\top} \pmod q \\ &= F_{\boldsymbol{v}}^{\top} \cdot \boldsymbol{s} + \left[\boldsymbol{x} \| \widetilde{\boldsymbol{R}}_1 \| \ldots \| \widetilde{\boldsymbol{R}}_t \right]^{\top} \pmod q \, . \end{aligned}$$

At this point, the short vector $\boldsymbol{e}_{\boldsymbol{v}} = \mathsf{SampleLeft}(F_{\boldsymbol{v}}, d_{\boldsymbol{v}}, \boldsymbol{u}, \tau_t)$ is computed by the decryption algorithm such that by Theorem 8, $F_{\boldsymbol{v}} \cdot \boldsymbol{e}_{\boldsymbol{v}} = \boldsymbol{u} \pmod q$. It follows that

$$\boldsymbol{e}_{\boldsymbol{v}}^{\top} \boldsymbol{c} = \boldsymbol{u}^{\top} \boldsymbol{s} + \boldsymbol{e}_{\boldsymbol{v}}^{\top} \left[\boldsymbol{x} \| \widetilde{\boldsymbol{R}}_1 \| \ldots \| \widetilde{\boldsymbol{R}}_t \right]^{\top} \pmod q \, .$$

Finally, the decryption algorithm computes:

$$\begin{aligned} z &= c' - \boldsymbol{e}_{\boldsymbol{v}}^{\top}\boldsymbol{c} \pmod q \\ &= \left(\boldsymbol{u}^{\top}\boldsymbol{s} + x + \mathfrak{m}\cdot\lfloor q/2\rceil\right) - \boldsymbol{u}^{\top}\boldsymbol{s} - \boldsymbol{e}_{\boldsymbol{v}}^{\top}\left[\boldsymbol{x}\|\widetilde{\boldsymbol{R}_1}\|\ldots\|\widetilde{\boldsymbol{R}_t}\right]^{\top} \pmod q \\ &= \mathfrak{m}\cdot\lfloor q/2\rceil + \underbrace{\left(x - \boldsymbol{e}_{\boldsymbol{v}}^{\top}\left[\boldsymbol{x}\|\widetilde{\boldsymbol{R}_1}\|\ldots\|\widetilde{\boldsymbol{R}_t}\right]^{\top}\right)}_{\text{noise term}} \pmod q\ . \end{aligned}$$

Thus, to have a successful decryption, it suffices to set the parameters so that with overwhelming probability,

$$\left| x - \boldsymbol{e}_{\boldsymbol{v}}^{\top}\left[\boldsymbol{x}\|\widetilde{\boldsymbol{R}_1}\|\ldots\|\widetilde{\boldsymbol{R}_t}\right]^{\top}\right| < \frac{q}{4}\ .$$

Let us write $\boldsymbol{e}_{\boldsymbol{v}} = [\boldsymbol{e}_{\boldsymbol{v},0}\|\boldsymbol{e}_{\boldsymbol{v},1}\|\ldots\|\boldsymbol{e}_{\boldsymbol{v},t}]$ with $\boldsymbol{e}_{\boldsymbol{v},i} \in \mathbb{Z}^m$ for $i = 0,\ldots,t$. Then the noise term can be rewritten as

$$x - \left(\boldsymbol{e}_{\boldsymbol{v},0} + \sum_{i\in[t]}\sum_{j\in[\mu]}\sum_{\gamma=0}^{k} v_{i,j,\gamma}\boldsymbol{R}_{i,j,\gamma}\boldsymbol{e}_{\boldsymbol{v},i}\right)^{\top}\boldsymbol{x}\ .$$

By Lemma 2, we have $\|\boldsymbol{e}_{\boldsymbol{v}}\| < \tau_t\sqrt{(t+1)m}$ with overwhelming probability. Moreover by Lemma 3, we have $\|\boldsymbol{R}_{i,j,\gamma}\cdot\boldsymbol{e}_{\boldsymbol{v},i}\| \le 12\sqrt{2m}\cdot\|\boldsymbol{e}_{\boldsymbol{v},i}\|$ with overwhelming probability, and since $v_{i,j,\gamma} \in [0, r-1]$ it follows that

$$\left\|\boldsymbol{e}_{\boldsymbol{v},0} + \sum_{i\in[t]}\sum_{j\in[\mu]}\sum_{\gamma=0}^{k} v_{i,j,\gamma}\boldsymbol{R}_{i,j,\gamma}\boldsymbol{e}_{\boldsymbol{v},i}\right\| = O(t\cdot\mu\cdot k\cdot r\cdot\sigma_t\cdot m)\ .$$

Finally, by Lemma 14, the error term has absolute value at most:

$$\left(q\alpha_t\cdot\omega(\sqrt{\log m}) + \sqrt{m}/2\right)\cdot O\left(t\cdot\mu\cdot k\cdot r\cdot\sigma_t\cdot m\right)\ .$$

3.4 Security Reduction

In this section we prove the following theorem.

Theorem 19. *If the decision-*$\mathsf{LWE}_{q,n,\chi}$ *problem is infeasible, then the predicate encryption scheme described above is weak attribute hiding-selective attribute secure.* ∎

Following [2,3], we define additional algorithms. These will not be used in the real scheme, but we need them in our proofs.

Sim.Setup$(1^n, \boldsymbol{\mu}, \boldsymbol{w}^\star)$. The algorithm choses random $\boldsymbol{A}$, $\boldsymbol{R}^\star_{i,j,k}$ and $\boldsymbol{u}$, it uses TrapGen to generate $\boldsymbol{B}^\star$ and defines $\boldsymbol{A}_{i,j,\gamma} \leftarrow \boldsymbol{A}\boldsymbol{R}^\star_{i,j\gamma} - r^\gamma w^\star_{i,j}\boldsymbol{B}^\star$. Specifically, on input a security parameter n, an hierarchical format of depth d $\boldsymbol{\mu} = (\ell, d; \mu_1,$

$\ldots, \mu_d)$, and a challenge vector $\boldsymbol{w}^\star = (\boldsymbol{w}_1^\star, \ldots, \boldsymbol{w}_d^\star)$, the algorithm generates public and secret parameters as follows: Choose random matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times m}$. For $i \in [d]$, $j \in [\mu_i]$ and $\gamma = 0, \ldots, k$, choose uniformly random matrices $\boldsymbol{R}_{i,j,\gamma}^\star \in \mathbb{Z}_q^{n \times m}$. Select a uniformly random vector $\boldsymbol{u} \in \mathbb{Z}_q^n$. Use algorithm $\mathsf{TrapGen}(q, n, m)$ to select a uniformly random $n \times m$-matrix $\boldsymbol{B}^\star \in \mathbb{Z}_q^{n \times m}$ with a basis $\boldsymbol{T}_{\boldsymbol{B}^\star} \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^\perp(\boldsymbol{B}^\star)$ such that $\|\widetilde{\boldsymbol{T}_{\boldsymbol{B}^\star}}\| \leq O(\sqrt{n \log q})$. For $i \in [d]$, $j \in [\mu_i]$ and $\gamma = 0, \ldots, k$, set $\boldsymbol{A}_{i,j,\gamma} \leftarrow \boldsymbol{A}\boldsymbol{R}_{i,j,\gamma}^\star - r^\gamma w_{i,j}^\star \boldsymbol{B}^\star$.

Output $mpk = (\boldsymbol{A}, \{\boldsymbol{A}_{i,j,\gamma}\}, \boldsymbol{u})$ and $msk = (\{\boldsymbol{R}_{i,j,\gamma}^\star\}, \boldsymbol{B}^\star, \boldsymbol{T}_{\boldsymbol{B}^\star})$.

Sim.Derive(mpk, d_v, v_t). Secret keys are now created by using the trapdoor $\boldsymbol{T}_{\boldsymbol{B}^\star}$, sampled by Sim.Setup, and the SampleBasisRight algorithm. Specifically, on input the master public key mpk, the secret key for the vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{t-1})$, and the vector $\boldsymbol{v}_t$, the algorithm generates a secret key for the vector $\boldsymbol{v'} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$ by constructing a short basis for $\Lambda_q^\perp(F_{\boldsymbol{v'}})$, as defined by Equation 2, by invoking $\boldsymbol{S} \leftarrow \mathsf{SampleBasisRight}(\boldsymbol{A}, \boldsymbol{B}_{\boldsymbol{v'}}^\star, \overline{\boldsymbol{R}^\star}, \boldsymbol{T}_{\boldsymbol{B}^\star}, \sigma_t)$. Output $d_{\boldsymbol{v'}} = \boldsymbol{S}$.

Sim.Enc$(mpk, \boldsymbol{w}, \mathfrak{m})$. The algorithm differs from Enc in the sense that it uses matrices $\boldsymbol{R}_{i,j,\gamma}^\star$ and $\boldsymbol{B}^\star$ instead of matrices $\boldsymbol{R}_{i,j,\gamma}$ and $\boldsymbol{B}$. Specifically, on input master public key mpk, vector $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t)$, and message $\mathfrak{m} \in \{0, 1\}$, the algorithm generates a ciphertext C as follows: Choose a uniformly random vector $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$, a noise vector $\boldsymbol{x} \leftarrow \overline{\Psi}_{\alpha_t}^m$ and a noise term $x \leftarrow \overline{\Psi}_{\alpha_t}$. Compute $\boldsymbol{c}_0 = \boldsymbol{A}^\top \boldsymbol{s} + \boldsymbol{x} \in \mathbb{Z}_q^m$. For $i \in [t]$, $j \in [\mu_i]$ and $\gamma = 0, \ldots, k$ compute $\boldsymbol{c}_{i,j,\gamma} = (\boldsymbol{A}_{i,j,\gamma} + r^\gamma w_{i,j} \boldsymbol{B}^\star)^\top \boldsymbol{s} + \boldsymbol{R}^{\star\top}_{i,j,\gamma} \boldsymbol{x} \in \mathbb{Z}_q^m$. Compute $c' = \boldsymbol{u}^\top \boldsymbol{s} + x + \mathfrak{m} \cdot \lfloor q/2 \rceil \in \mathbb{Z}_q$. Output $C = (\boldsymbol{c}_0, \{\boldsymbol{c}_{i,j,\gamma}\}, c')$.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of 6 games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

$\mathbf{Game_0}$. $\mathcal{C}$ runs the Setup algorithm, answers $\mathcal{A}$'s secret key queries using the Derive algorithm, and generates the challenge ciphertext using the Enc with vector $\boldsymbol{w}_0$ and message $\mathfrak{m}_0$.

$\mathbf{Game_1}$. In this game $\mathcal{C}$ uses the simulation algorithms. Specifically, $\mathcal{C}$ runs the Sim.Setup algorithm with $\boldsymbol{w}^\star = \boldsymbol{w}_0$, answers $\mathcal{A}$'s secret key queries using the Sim.Derive algorithm, and generates the challenge ciphertext using the Sim.Enc with vector $\boldsymbol{w}_0$ and message $\mathfrak{m}_0$.

$\mathbf{Game_2}$. It is the same as the $\mathbf{Game_1}$ except that the challenge ciphertext is randomly chosen from the ciphertext space.

$\mathbf{Game_3}$. It is the same as the $\mathbf{Game_2}$ except that $\mathcal{C}$ runs the Sim.Setup algorithm with $\boldsymbol{w}^\star = \boldsymbol{w}_1$.

$\mathbf{Game_4}$. It is the same as the $\mathbf{Game_3}$ except that $\mathcal{C}$ generates the challenge ciphertext using the Sim.Enc with vector $\boldsymbol{w}_1$ and message $\mathfrak{m}_1$.

$\mathbf{Game_5}$. $\mathcal{C}$ runs the Setup algorithm, answers $\mathcal{A}$'s secret key queries using the Derive algorithm, and generates the challenge ciphertext using the Enc with vector $\boldsymbol{w}_1$ and message $\mathfrak{m}_1$.

We defer the proof that, for $i = 0, \ldots, 4$, $\mathbf{Game_i}$ is indistinguishable form $\mathbf{Game_{i+1}}$ under the appropriate assumptions, to the full version of this paper.

Parameters. From the previous sections we can extract the parameters required for correctness and security of the system.

- We need to ensure that for each $t \in [d]$, correctness holds. Specifically, Lemma 18 requires $q/\log q = \Omega(\sigma_t \cdot \mu \cdot \frac{r}{\log r} \cdot m^{3/2})$ and $\alpha_t \leq (t \cdot \log q \cdot \sigma_t \cdot \mu \cdot \frac{r}{\log r} \cdot m \cdot \omega(\sqrt{\log m}))^{-1}$.
- By Theorem 6, algorithm $\mathsf{TrapGen}$ requires $q > 2$ and $m > 6n \lg q$ to work.
- By Corollary 16, to have algorithm $\mathsf{SampleBasisLeft}$ working correctly in the Derive algorithm, we need for each $t \in [d]$, $\sigma_t > \|\widetilde{d_{\boldsymbol{v}}}\| \cdot \omega(\sqrt{\log((t+1)m)})$. Thus, we have $\sigma_t > \sigma_{\mathrm{TG}} \cdot \omega((\log m)^{t/2})$.
- By Corollary 17, to have algorithm $\mathsf{SampleBasisRight}$ working correctly in the $\mathsf{Sim.Derive}$ algorithm, we need for each $t \in [d]$, $\sigma_t > \|\widetilde{\boldsymbol{T}_{\boldsymbol{B}}}\| \cdot s_{\overline{\boldsymbol{R}}} \cdot \omega(\sqrt{\log((t+1)m)})$. Thus, by Theorem 6, $\|\widetilde{\boldsymbol{T}_{\boldsymbol{B}}}\| < \sigma_{\mathrm{TG}}$ and, by Lemma 3, $s_{\overline{\boldsymbol{R}}} = \|\overline{\boldsymbol{R}}\| = O(t \cdot \mu \cdot (\log_r q + 1) \cdot \sqrt{(t+1)m})$ due the particular structure of $\overline{\boldsymbol{R}}$, where $\mu = \max_{i \in [d]} \mu_i$. Thus, $\sigma_t > O(\sqrt{n \log q}) \cdot O(\mu \cdot (\log_r q + 1) \cdot \sqrt{(t+1)m}) \cdot \omega(\sqrt{(t+1)m})$.
- Regev's reduction must apply: $q > 2\sqrt{n}/\alpha_t$

3.5 Wrapping Up

Proof of Theorem 19. From the previous sections and lemmata, we have shown that our hierarchical inner-product encryption is weak attribute hiding-selective attribute secure assuming decision-$\mathsf{LWE}_{q,n,\chi}$ problem is infeasible.

4 Application

4.1 Identity-Based Encryption with Wildcards

One of the main applications of IBE and HIBE schemes is email encryption, where users can encrypt a message to the owner of the email address without having to obtain a certified copy of the owner's public key first. Motivated by the fact that many email addresses correspond to groups of users rather than single individuals, Abdalla *et al.* [1] introduced the concept of identity-based cryptography with wildcards (WIBE). In a WIBE scheme, decryption keys are issued exactly as in a standard HIBE scheme and the main difference lies in the encryption process. More specifically, in a WIBE scheme, the sender can encrypt messages to more general *patterns* consisting of identity strings and *wildcards* so that any identity matching the given pattern can decrypt.

Next we show how to convert any HIPE scheme in a WIBE one by using an encoding first introduced in [13]. This let us to obtain the first WIBE scheme based on lattice assumptions.

Let us start with some notation. A pattern is described by a vector $P = (P_1, \ldots, P_\ell) \in (\{0,1\}^\star \bigcup \{\star\})^\ell$, where $\star$ is a special wildcard symbol. We say that identity $ID = (ID_1, \ldots, ID_{\ell'})$ matches P, denoted $ID \in_\star P$, if and only if $\ell' \leq \ell$ and for all $i = 1, \ldots, \ell'$ we have that $ID_i = P_i$ or $P_i = \star$. Note that under this definition, any ancestor of a matching identity is also a matching identity. This is reasonable for

our purposes because any ancestor can derive the secret key of a matching descendant identity anyway. If $P = (P_1, \ldots, P_\ell)$ is a pattern, then we define $\mathrm{W}(P)$ to be the set of wildcard positions in P, i.e. $\mathrm{W}(P) = \{1 \leq i \leq \ell \ : \ P_i = *\}$. Formally, a WIBE scheme is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Derive}, \mathsf{Enc}, \mathsf{Dec})$ providing the following functionality. The Setup and Derive algorithms behave exactly as those of a HIPE scheme. To create a ciphertext of a message $\mathfrak{m} \in \{0,1\}^*$ intended for all identities matching pattern P, the sender computes $C \leftarrow \mathsf{Enc}(mpk, P, \mathfrak{m})$. Any of the intended recipients $ID \in_* P$ can decrypt the ciphertext using its own decryption key as $\mathfrak{m} \leftarrow \mathsf{Dec}(d_{ID}, C)$.

Let $\mathsf{HIPE} = (\mathsf{Setup}_\mathsf{H}, \mathsf{Derive}_\mathsf{H}, \mathsf{Enc}_\mathsf{H}, \mathsf{Dec}_\mathsf{H})$ be a Hierarchical inner-product encryption. We can construct the scheme $\mathsf{WIBE} = (\mathsf{Setup}_\mathsf{W}, \mathsf{Derive}_\mathsf{W}, \mathsf{Enc}_\mathsf{W}, \mathsf{Dec}_\mathsf{W})$ as follows:

$\mathsf{Setup}_\mathsf{W}(1^\lambda, 1^\ell)$. The algorithm returns the output of $\mathsf{Setup}_\mathsf{H}(1^\lambda, \boldsymbol{\mu} = (2\ell, \ell; (\mu_i = 2)_{i\in[\ell]}))$. So the hierarchy $\boldsymbol{\mu}$ is of depth ℓ and each level has dimension 2.

$\mathsf{Derive}_\mathsf{W}(msk, ID)$. For a pattern $ID = (ID_1, \ldots, ID_\ell)$, the key generation algorithm constructs vector $\boldsymbol{y} \in \Sigma$ by setting, for each $i \in [\ell]$, $\boldsymbol{y}_i = (1, P_i)$. We denote this transformation by $\boldsymbol{y} = \mathsf{KEncode}(ID)$. Then the key generation algorithm returns $d_P = \mathsf{Derive}_\mathsf{H}(msk, \boldsymbol{y})$.

$\mathsf{Enc}_\mathsf{W}(mpk, P)$. The algorithm constructs vector $\boldsymbol{x} \in \Sigma$ in the following way: For each $i \in [\ell]$ the algorithms sets $\boldsymbol{x}_i = (-r_i \cdot P_i, r_i)$ if $P_i \neq *$, $\boldsymbol{x}_i = (0,0)$ otherwise. We denote this transformation by $\boldsymbol{x} = \mathsf{CEncode}(\boldsymbol{z})$. Then the encryption algorithm returns $C = \mathsf{Enc}_\mathsf{H}(mpk, \boldsymbol{x})$.

$\mathsf{Dec}_\mathsf{W}(d_P, C)$. The algorithm returns the output of $\mathsf{Dec}_\mathsf{H}(d_P, C)$.

Correctness. Correctness follows from the observation that for identity ID and pattern P, we have that $f_{\mathsf{KEncode}(ID)}(\mathsf{CEncode}(P)) = 1$ if and only if $ID \in_* P$.

Security. Let $\mathcal{A}$ be an adversary for WIBE that tries to break the scheme for an hierarchy of depth ℓ and consider the following adversary $\mathcal{B}$ for HIPE that uses $\mathcal{A}$ as a subroutine and tries to break a HIPE with hierarchy format $\boldsymbol{\mu} = (2\ell, \ell; (\mu_i = 2i)_{i\in[\ell]})$ by interacting with challenger $\mathcal{C}$. $\mathcal{B}$ receives a mpk for HIPE and passes it to $\mathcal{A}$. Whenever $\mathcal{A}$ asks for the key for identity ID, $\mathcal{B}$ constructs $\boldsymbol{y} = \mathsf{KEncode}(P)$ and asks $\mathcal{C}$ for a key $d_{\boldsymbol{y}}$ for $\boldsymbol{y}$ and returns it to $\mathcal{A}$. When $\mathcal{A}$ asks for a challenge ciphertext by providing $(\mathfrak{m}_0, P_0^\star)$ and $(\mathfrak{m}_1, P_1^\star)$, $\mathcal{B}$ simply computes $\boldsymbol{x}_0 = \mathsf{CEncode}(P_0^\star)$ and $\boldsymbol{x}_1 = \mathsf{CEncode}(P_1^\star)$ and gives the pair $(\mathfrak{m}_0, \boldsymbol{x}_0), (\mathfrak{m}_1, \boldsymbol{x}_1)$ to $\mathcal{C}$. $\mathcal{B}$ then returns the challenge ciphertext obtained from $\mathcal{C}$ to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs $\mathcal{A}$'s guess. Notice that, $\mathcal{B}$'s simulation is perfect. Indeed, we have that if for all $\mathcal{A}$'s queries satisfy the game constraint, then all $\mathcal{B}$'s queries have the same property. Thus $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s.

4.2 Chosen-Ciphertext Security

As we have seen in the previous section, given an HIPE scheme is possible to construct a WIBE scheme. Thus, to apply the techniques of [6] to obtain an ℓ-level HIPE scheme secure against chosen-ciphertext attacks we will face the same issues faced by [1].

Thus, following [1], we show that we may use a IND-wAH-sID-HIPE-CPA-secure HIPE of depth $2d + 2$ and a strongly unforgeable signature scheme

$(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Verify})$ to construct an IND-wAH-sID-HIPE-CCA-secure HIPE of depth d. We adapt the encoding function Encode defined in [1] to the HIPE case in the following way: For a HIPE scheme for hierarchical format $\boldsymbol{\mu} = (\ell, d; \mu_1, \ldots, \mu_d)$, we define two encoding functions, one to encode secret keys and one to encode ciphertext. Specifically, for any two values a and b in $\mathbb{Z}_N^\star$ such that $a \neq b$, the encode function for secret keys $\mathsf{KEncode}$ works as follow: $\mathsf{KEncode}(\boldsymbol{v}) = ((1,a), \boldsymbol{v}_1, \ldots, (1,a), \boldsymbol{v}_t)$, for any vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$ with $t \leq d$ and $\mathsf{KEncode}(\boldsymbol{v}, vk) = ((1,a), \boldsymbol{v}_1, \ldots, (1,a), \boldsymbol{v}_t, (1,b), (1, vk))$. On the other hand, the encode function for ciphertext $\mathsf{CEncode}$ works as follow: $\mathsf{CEncode}(\boldsymbol{w}) = ((a,-1), \boldsymbol{w}_1, \ldots, (a,-1), \boldsymbol{w}_t)$, for any vector $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t)$ with $t \leq d$ and $\mathsf{CEncode}(\boldsymbol{w}, vk) = ((a,-1), \boldsymbol{w}_1, \ldots, (a,-1), \boldsymbol{w}_t, (b,-1), (vk,-1))$.

Construction. Given a HIPE scheme $\mathsf{HIPE} = (\mathsf{Setup}, \mathsf{Derive}, \mathsf{Enc}, \mathsf{Dec})$ for hierarchical format $\boldsymbol{\mu} = (\ell, 2d+2; 2, \mu_1, 2, \mu_2, \ldots, 2, \mu_d, 2, 2)$, consider the following HIPE scheme $\mathsf{HIPE}' = (\mathsf{Setup}', \mathsf{Derive}', \mathsf{Enc}', \mathsf{Dec}')$ for hierarchical format $\boldsymbol{\mu} = (\ell, d; \mu_1, \mu_2, \ldots, \mu_d)$.

Key Derivation. The secret key for vector $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$, with $t \leq d$ under HIPE' is the secret key corresponding to identity $\mathsf{KEncode}(\boldsymbol{v})$ under HIPE.

Encryption. To encrypt a message $\mathfrak{m}$ under a vector $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_t)$ and using a master public key mpk, the following steps are performed: First, generate a signature key pair $(sk, vk) \xleftarrow{\$} \mathsf{SigGen}$. Then compute $C \xleftarrow{\$} \mathsf{Enc}(mpk, \mathsf{CEncode}(\boldsymbol{w}, vk), \mathfrak{m})$ and $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, C)$. The final ciphertext is (vk, C, σ).

Decryption. To decrypt a ciphertext (vk, C, σ) using a private key $d_{\boldsymbol{v}}$ for a vector $\boldsymbol{v}$, first check that $\mathsf{Verify}(vk, C, \sigma) = \texttt{valid}$. If not, output $\perp$. Otherwise, compute $d = \mathsf{Derive}(d_{\boldsymbol{v}}, ((1,b), (1, vk)))$ and output $\mathsf{Dec}(d, C)$. Note that in this case d is the decryption for the identity $\mathsf{KEncode}(\boldsymbol{v}, vk)$ in HIPE.

Proof Sketch. Let $\mathcal{A}$ be an IND-wAH-sID-HIPE-CCA adversary against the HIPE' scheme. Then there exists an IND-wAH-sID-HIPE-CPA attacker $\mathcal{C}$ against HIPE that uses $\mathcal{A}$ as a subroutine. $\mathcal{C}$ can simulate $\mathcal{A}$'s environment in a straight forward way. Then $\mathcal{C}$ wins the game whenever $\mathcal{A}$ does as long as $\mathcal{C}$ does not make any illegal key derivation queries. We will argue this fact briefly. First consider the queries that $\mathcal{C}$ makes to respond to $\mathcal{A}$'s key derivation query $\boldsymbol{v}$. Let $\boldsymbol{v'} = \mathsf{KEncode}(\boldsymbol{v})$ and let $\boldsymbol{w}^{'\star} = \mathsf{CEncode}(\boldsymbol{w}^\star, vk^\star)$. We have the following:

1. If $|\boldsymbol{v'}| > |\boldsymbol{w}^{'\star}|$ then $f_{\boldsymbol{v'}}(\boldsymbol{w}^{'\star}) = 0$.
2. If $|\boldsymbol{v}| = |\boldsymbol{w}^{'\star}|$ then still $f_{\boldsymbol{v'}}(\boldsymbol{w}^{'\star}) = 0$ because $\boldsymbol{v'}$ and $\boldsymbol{w}^{'\star}$ are different on the next to last level ($\boldsymbol{v'}$ contains $(1,a)$ there, while $\boldsymbol{w}^{'\star}$ contains a $(b,-1)$ and they are not orthogonal).
3. If $|\boldsymbol{v'}| < |\boldsymbol{w}^{'\star}|$ then the only way to have $f_{\boldsymbol{v'}}(\boldsymbol{w}^{'\star}) = 1$ is if also $f_{\boldsymbol{v}}(\boldsymbol{w}^\star) = 1$, which are illegal queries in $\mathcal{A}$'s game as well.

Second, consider the key derivation queries that $\mathcal{C}$ makes in order to respond to $\mathcal{A}$'s decryption queries. If $\mathcal{A}$ makes decryption query $(\boldsymbol{v}, (vk, C, \sigma))$, then $\mathcal{C}$ makes a key derivation query for $\boldsymbol{v'} = \mathsf{KEncode}(\boldsymbol{v}, vk)$. Let $\boldsymbol{w}^{'\star} = \mathsf{CEncode}(\boldsymbol{w}^\star, vk^\star)$. Then we have two cases:

1. If $vk \neq vk^\star$ then $f_{\boldsymbol{v}'}(\boldsymbol{w}'^\star) = 0$ either because $\boldsymbol{v}$ has a $(1, a)$ where $\boldsymbol{w}'^\star$ has a $(b, -1)$, or because they differ on the last level.
2. If $vk = vk^\star$, then we have the following three sub-cases:
   (a) If $|\boldsymbol{v}'| > |\boldsymbol{w}'^\star|$ then $f_{\boldsymbol{v}'}(\boldsymbol{w}'^\star) = 0$.
   (b) If $|\boldsymbol{v}'| < |\boldsymbol{w}'^\star|$ then still $f_{\boldsymbol{v}'}(\boldsymbol{w}'^\star) = 0$ because $\boldsymbol{v}'$ and $\boldsymbol{w}'^\star$ are different on the next to last level ($\boldsymbol{v}'$ contains $(1, a)$ there, while $\boldsymbol{w}'^\star$ contains a $(b, -1)$ and they are not orthogonal).
   (c) If $|\boldsymbol{v}| = |\boldsymbol{w}'^\star|$ then the only way to have $f_{\boldsymbol{v}'}(\boldsymbol{w}'^\star) = 1$ is if also $f_{\boldsymbol{v}}(\boldsymbol{w}) = 1$ but this case can be proved to have negligible probability under the one-time security of the signature scheme.

Acknowledgments. This work was supported by the French ANR-09-VERS-016 BEST Project, by the Brazilian CAPES Foundation, and by the European Commission through the ICT Program under Contract ICT-2007-216676 ECRYPT II and the FP7-ICT-2011-EU-Brazil Program under Contract 288349 SecFuNet.

References

1. Abdalla, M., Catalano, D., Dent, A.W., Malone-Lee, J., Neven, G., Smart, N.P.: Identity-Based Encryption Gone Wild. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 300–311. Springer, Heidelberg (2006)
2. Agrawal, S., Boneh, D., Boyen, X.: Efficient Lattice (H)IBE in the Standard Model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 553–572. Springer, Heidelberg (2010)
3. Agrawal, S., Freeman, D.M., Vaikuntanathan, V.: Functional Encryption for Inner Product Predicates from Learning with Errors. In: Lee, D.H. (ed.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 21–40. Springer, Heidelberg (2011)
4. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: ACM STOC Annual ACM Symposium on Theory of Computing, pp. 99–108. ACM Press (May 1996)
5. Alwen, J., Peikert, C.: Generating shorter bases for hard random lattices. In: STACS 2009, pp. 75–86 (2009)
6. Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM Journal on Computing 36(5), 1301–1328 (2007)
7. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public Key Encryption with Keyword Search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
8. Boneh, D., Sahai, A., Waters, B.: Functional Encryption: Definitions and Challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011)
9. Boneh, D., Waters, B.: Conjunctive, Subset, and Range Queries on Encrypted Data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
10. 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)
11. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC Annual ACM Symposium on Theory of Computing, pp. 197–206. ACM Press (May 2008)

12. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Juels, A., Wright, R.N., Vimercati, S. (eds.) ACM CCS 2006: 13th Conference on Computer and Communications Security, pp. 89–98. ACM Press (October/November 2006)
13. Katz, J., Sahai, A., Waters, B.: Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (2008)
14. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010)
15. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. In: 45th FOCS Annual Symposium on Foundations of Computer Science, pp. 372–381. IEEE Computer Society Press (October 2004)
16. Okamoto, T., Takashima, K.: Hierarchical Predicate Encryption for Inner-Products. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 214–231. Springer, Heidelberg (2009)
17. Okamoto, T., Takashima, K.: Adaptively Attribute-Hiding (Hierarchical) Inner Product Encryption. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 591–608. Springer, Heidelberg (2012)
18. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: Mitzenmacher, M. (ed.) 41st ACM STOC Annual ACM Symposium on Theory of Computing, pp. 333–342. ACM Press (May/June 2009)
19. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC Annual ACM Symposium on Theory of Computing, pp. 84–93. ACM Press (May 2005)
20. Sahai, A., Waters, B.: Fuzzy Identity-Based Encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)

Towards Efficient Arithmetic for Lattice-Based Cryptography on Reconfigurable Hardware

Thomas Pöppelmann and Tim Güneysu*

Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany

Abstract. In recent years lattice-based cryptography has emerged as quantum secure and theoretically elegant alternative to classical cryptographic schemes (like ECC or RSA). In addition to that, lattices are a versatile tool and play an important role in the development of efficient fully or somewhat homomorphic encryption (SHE/FHE) schemes. In practice, ideal lattices defined in the polynomial ring $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$ allow the reduction of the generally very large key sizes of lattice constructions. Another advantage of ideal lattices is that polynomial multiplication is a basic operation that has, in theory, only quasi-linear time complexity of $\mathcal{O}(n \log n)$ in $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$. However, few is known about the practical performance of the FFT in this specific application domain and whether it is really an alternative. In this work we make a first step towards efficient FFT-based arithmetic for lattice-based cryptography and show that the FFT can be implemented efficiently on reconfigurable hardware. We give instantiations of recently proposed parameter sets for homomorphic and public-key encryption. In a generic setting we are able to multiply polynomials with up to 4096 coefficients and a 17-bit prime in less than 0.5 milliseconds. For a parameter set of a SHE scheme (n=1024,p=1061093377) our implementation performs 9063 polynomial multiplications per second on a mid-range Spartan-6.

Keywords: Lattice-Based Cryptography, Ideal Lattices, FFT, NTT, FPGA Implementation.

1 Introduction

Currently used cryptosystems rely on similar number theoretical problems such as the hardness of factoring (RSA) or the elliptic curve discrete logarithm problem (ECC). Unfortunately, these do not hold any more in case it becomes feasible to build a large enough quantum computer [52]. Even without quantum computers it may occur that improvements of cryptanalytic algorithms heavily impact the security of the currently used schemes or raise their required security parameters beyond practicability. Therefore, it is necessary to investigate practical and efficient alternative cryptosystems on relevant architectures (hardware/software) that can withstand quantum attacks [12].

* This work was partially supported by European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 139–158, 2012.

Promising candidates to satisfy this need are lattice-based cryptosystems. They rely on well-studied problems for which no efficient quantum algorithms are known and also enable elegant security proofs and worst-case to average-case reductions. However, for long lattice-based cryptography has been considered secure only for large parameter sets beyond practicability (e.g., a key size around one megabyte [42]) and required complex operations like multiplication of large matrices. This has changed since the introduction of cyclic [41] and ideal lattices [35] which enable the construction of a great variety of theoretically sound and also efficient cryptosystems.

Besides being considered as a replacement for classical schemes, (ideal) lattices are also employed for new primitives like fully or somewhat homomorphic encryption (SHE/FHE) that have the potential to secure cloud computing. The first fully homomorphic encryption system has been proposed by Gentry in 2009 [24] and with his work, research in SHE/FHE gathered momentum due to the great variety of applications and new ideas. However, efficiency is still far from being practical and a lot of research focuses on performance improvements for SHE/FHE [25,11,43].

When considering the runtime performance of proposed ideal lattice-based cryptosystems, the most common and also most expensive operation is polynomial multiplication in $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$. This is also one reason for the often claimed asymptotic speed advantage of lattice-based schemes over number theoretical constructions, as polynomial multiplication is equivalent to the negacyclic or negative-wrapped convolution product which can be efficiently computed in quasi-linear time $\mathcal{O}(n \log n)$[1] using the Fast Fourier Transform (FFT). However, this notation omits constants and terms of lower order and in practice it is not always clear if simpler algorithms like schoolbook multiplication or Karatsuba [32] perform better on a specific platform.

In case of software implementations it is usually easy to determine which polynomial multiplication algorithm yields the best performance for a target architecture by running a few experiments with the desired parameter sets. Especially, as the simple algorithms are easy to implement and the more evolved ones are already available and heavily optimized in the NTL [53] library of Victor Shoup. However, for hardware this is different as only very few results are available [29,26] which consume lots of FPGA resources and do not target larger parameter sets needed for SHE/FHE.

Some examples of previous implementations of polynomial multiplication not designed for lattice-based cryptography are [56,14,7] but they mostly deal with Galois Fields $GF(2^n)$. Other implementations are only optimized for the computation of approximate results due to the complex floating-point arithmetic. This lack of reference implementations may prevent the usage of lattice-based cryptography in hardware. We therefore present a versatile implementation of arithmetic for lattice-based cryptography and hope that this will allow designers to focus more on the specific nature of a scheme than on the tedious implementation of the underlying FFT arithmetic.

[1] All logarithms in this paper are of base two.

Our Contribution. In this work we present an adaptable and extensible FPGA implementation of polynomial multiplication based on the FFT in $\mathbb{Z}_p$. All arithmetic operations are specifically optimized for the ring $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$ which is heavily used in ideal lattice-based cryptography, especially homomorphic encryption. Our implementation supports a broad range of parameters and we give optimized variants for previously proposed parameter sets for public key encryption and SHE. We show that the FFT seems to be the method of choice for future implementations and that homomorphic encryption can benefit from FPGA hardware accelerators. Our design is scalable, has small area consumption on a low-cost FPGA and offers decent performance compared to general purpose computers. We will make our source files, test-cases and documentation available in order to allow other researchers to use and evaluate our implementation.[2]

Outline. This work is structured as follows. At first we give a short introduction to lattice-based cryptography and introduce different methods for polynomial multiplication and outline their theoretical complexity in the next section. We then discuss several design options in Section 4 and detail our implementations of these algorithms on reconfigurable hardware in Section 5. We finally give considerations on the expected runtime and present performance results for a broad range of parameters in Section 6. We finish this paper with future work and a conclusion.

2 Ideal Lattice-Based Cryptography

Lattice-based cryptography has become more and more important in the last years starting with the seminal result by Ajtai [2] who proved a worst-case to average-case reduction between several lattice problems. In addition to that, no efficient quantum algorithms solving lattice problems are currently known. The two most commonly used average-case lattice problems are the short integer solution (SIS) [2] and the learning with errors (LWE) [48] problem. Both can be shown to be as hard as solving other lattice problems in the worst-case, e.g., the (gap) shortest vector problem (SVP).

However, while schemes based on SIS or LWE admit very appealing theoretical properties they are not efficient enough for practical usage, especially as in most schemes the key-size grows (at least) quadratically in $\mathcal{O}(n^2)$ for the security parameter n [42]. This problem can be mitigated by introducing algebraic structure into the previously completely randomly generated lattice. This led to the definition of cyclic [41] and the more generalized ideal lattices [35] which correspond to ideals in the ring $\mathbb{Z}[x]/\langle f\rangle$ for some irreducible polynomial f of degree n. They proved to be a valuable tool for the construction of signature schemes [37,38], collision resistant hash functions [39,3], public key encryption schemes [36,54], identification schemes [34] as well as FHE [24] and SHE [43]. While certain properties can be established for various rings, in most cases the ring $R = \mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$ is used with $f(x) = x^n + 1$, n being a power of 2 and p being a prime such that $p \equiv 1 \bmod 2n$ (e.g., [37,39,43,54])

[2] See our web page at http://www.sha.rub.de/research/projects/lattice/

In the ring variant of the previously mentioned decisional LWE problem the attacker has to decide whether the samples $(a_1, t_1), \ldots, (a_m, t_m) \in R \times R$ are chosen completely random or whether each $t_i = a_i s + e_i$ with $s, e_1, \ldots, e_m$ having small coefficients from a Gaussian distribution. The pseudo-randomness of the ring-LWE distribution can then directly be applied to construct a semantically secure public-key cryptosystem. When defining this cryptosystem in $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$ it is also possible to compute the polynomial product of the noisy coefficients of the ring-LWE problem efficiently by using the Fast Fourier Transform (FFT) in the field $\mathbb{Z}_p$. This gives lattice-based cryptosystems the advantage that for most cases the runtime is quasi-linear in $\mathcal{O}(n \log n)$ and an increase of the security parameter affects the efficiency far less than in other proposals [36].

For practical applications theoretical proofs of security are an important foundation but the choice of parameters is much more relevant. If the parameters are chosen in a way that the underlying problem, to which the security of a scheme is reduced, is actually easy, then not much is gained. Therefore, as of now lattice-based cryptosystems have to proof that they allow efficient and fast implementations for reasonable parameters that yield practical key and ciphertext sizes while maintaining quantum security and reducibility to hard lattice problems. As ideal lattice cryptosystems allow actually practical or at least not totally out of reach parameters, several researchers have suggested parameters for their schemes. The general approach for selecting parameters used by [37] and [42] is based on the experiments by Gama and Nguyen [22]. However, other works [50,33,13] extends these method and provide frameworks for the derivation of parameters for several security levels. These security levels allow the direct calculation of the key and plaintext size.

Concerning implementations, lattice-based cryptography has very recently received attention by researchers. For example Györfi et al. proposed a high-speed implementation of the SWIFFT and SWIFFTX hash function on reconfigurable hardware [29] which uses the FFT/NTT of order 64 over $\mathbb{Z}_{257}$. This hardware implementation is considerably faster (claimed 16× speed-up) compared to a SIMD software implementation provided by the designers of SWIFFT(X) in their submission to the NIST competition [39]. A software implementations of an LWE-based public key encryption systems is described in [21] and an implementation in hardware for power-constrained RFID applications is presented in [61]. Very recent work by Göttert et al. [26] covers a software implementation as well as an optimized realization of the scheme with the help of the FFT/NTT on reconfigurable hardware. An efficient signature scheme implemented on reconfigurable hardware is presented in [27]. However, the number of available implementations is still limited compared to the high number of proposed lattice-based cryptosystems. Due to the large variety in possible parameter sets it is still a challenge to determine how fast and efficient lattice-based cryptography can be on general purpose processors, micro-controllers and also reconfigurable hardware. A notable exception to this is the ad-hoc designed NTRU encryption system (defined in $\mathbb{Z}_p[\mathbf{x}]/\langle x^n - 1\rangle$) [30] for which several implementation results and also two standards have been published over the years (e.g., [31,4,5]).

3 Efficient Polynomial Multiplication in $\mathbb{Z}_p[\mathrm{x}]/\langle x^n + 1\rangle$

In this section we shortly revisit well-known polynomial multiplication algorithms applicable for ideal lattice-based cryptosystems (see [9] for a comprehensive treatment). As mentioned in Section 2, the currently used ideal lattices are equivalently characterized as ideals of the ring of modular polynomials $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$ with n integer coefficients such that $f(x) = f_0 + f_1 x + f_2 x^2 + ... + f_{n-1} x^{n-1} \in \mathbb{Z}_p[x]$ [49]. Two elements in these rings can be easily added coefficient-wise with a complexity of $\mathcal{O}(n)$. However, multiplication is much more complicated and for $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$ the product $c(x) = a(x) \cdot b(x)$ can be computed by considering the special rule that $x^n \equiv -1$. This leads to

$$\mathbf{ab} = \left[\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j x^{i+j} \right] \bmod \langle x^n + 1\rangle = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (-1)^{\left\lfloor \frac{i+j}{n} \right\rfloor} a_i b_j x^{i+j \bmod n}.$$

This classical "schoolbook" algorithm has a quadratic complexity $\mathcal{O}(n^2)$ and generally requires n^2 multiplications and $(n-1)^2$ additions or subtractions [57]. The modulus n operation can be performed efficiently by a right shift when n is a power of two and $(-1)^{\left\lfloor \frac{i+j}{n} \right\rfloor}$ is 1 for $i + j < n$ and -1 otherwise as $i + j \leq 2n - 2$. The complexity of polynomial multiplication can be lowered to $\mathcal{O}(n^{\log(3)})$ with the Karatsuba method [32] but due to its recursive structure the original algorithm is not supposed to perform well on hardware (see [17] for implementation decisions regarding a Karatsuba multiplier).

3.1 The Number Theoretic Transform

A quasi-linear complexity of $\mathcal{O}(n \log n)$ for polynomial multiplication can be achieved with the Fast Fourier Transform (FFT) popularized by Cooley and Turkey [15]. One main application is integer multiplication (e.g., Schönhage and Strassen [51]) and signal processing [10]. The Number Theoretic Transform (NTT) [46] is essentially an FFT defined in a finite field or ring so that no inaccurate floating point or complex arithmetic is needed. For a given primitive n-th root of unity ω, the generic forward $\mathrm{NTT}_\omega(a)$ of a sequence $\{a_0, .., a_{n-1}\}$ to $\{A_0, \ldots, A_{n-1}\}$ with elements in $\mathbb{Z}_p$ and length n is defined as

$$A_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \bmod p, \;\; i = 0, 1, ..., n-1 \tag{1}$$

with the inverse $\mathrm{NTT}_\omega^{-1}(A)$ being denoted as

$$a_i = n^{-1} \sum_{j=0}^{n-1} A_j \omega^{-ij} \bmod p, \;\; i = 0, 1, ..., n-1. \tag{2}$$

The NTT exists if and only if the block length n divides $q - 1$ for every prime factor q of p [10]. The value ω is a primitive n-th root of unity if (1) $\omega^n = 1 \bmod p$

and (2) $\omega^{n/q} - 1 \neq 0 \bmod p$ for any prime divisor q of n [7]. For p being a prime and n being a power of two, the NTT can be computed with a simple algorithm with time complexity $\mathcal{O}(n \log n)$ when $n \bmod (p-1) \equiv 1$.

Theorem 1 (Convolution Theorem [59]). *Let ω be a primitive $2n$-th root of unity in $\mathbb{Z}_p$. Let $a = (a_0, ..., a_{n-1})$ and $b = (b_0, ..., b_{n-1})$ be vectors of length n with elements in $\mathbb{Z}_p$ and $\tilde{a} = (a_0, ..., a_{n-1}, 0, ..., 0)$,$\tilde{b} = (b_0, ..., b_{n-1}, 0, ..., 0)$ the corresponding vectors of length $2n$, where the trailing components have been filled with zeros. With $\circ$ meaning component-wise multiplication then $a \cdot b = NTT_{\omega}^{-1}(NTT_{\omega}(\tilde{a}) \circ NTT_{\omega}(\tilde{b}))$.*

By Theorem 1 we can now multiply arbitrary polynomials represented as vectors and later (if necessary) reduce them with a reduction polynomial of our choice. However, as we have to append n zeros, the length of the input sequence to the transform doubles.

Theorem 2 (Wrapped Convolution [59]). *Let ω be a primitive n-th root of unity in $\mathbb{Z}_p$ and $\psi^2 = \omega$. Let $a = (a_0, ..., a_{n-1})$ and $b = (b_0, ..., b_{n-1})$ be vectors of length n with elements in $\mathbb{Z}_p$.*

1. *The positive wrapped convolution of a and b is $NTT_w^{-1}(NTT_w(a) \circ NTT_w(b))$.*
2. *Let $d = (d_0, ..., d_{n-1})$ be the negative wrapped convolution of a and b. Let $\bar{a}, \bar{b}$ and $\bar{d}$ be defined as $(a_0, \psi a_1, ..., \psi^{n-1} a_{n-1})$, $(b_0, \psi b_1, ..., \psi^{n-1} b_{n-1})$, and $(d_0, \psi d_1, ..., \psi^{n-1} d_{n-1})$. Then $\bar{d} = NTT_w^{-1}(NTT_w(\bar{a}) \circ NTT_w(\bar{b}))$.*

In order to avoid the doubling of the input length of the NTT as described in Theorem 1 we can directly perform a positive wrapped convolution for arithmetic in $\mathbb{Z}_p[\mathbf{x}]/\langle x^n - 1 \rangle$ and use the negative wrapped convolution for $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1 \rangle$. The latter case is already exploited in practice for a lattice-based hash function (see SWIFFT [39]) and particularly interesting as we are getting the necessary reduction by the polynomial $x^n + 1$ for "free" and can work with a transform length that is equal to the number of polynomial coefficients. The only restriction is that we have to find an n-th root of unity ω and its modular square root ψ such that $\psi^2 \equiv \omega \bmod p$. As a consequence, when p is a prime and n a power of two, the negative wrapped convolution approach is only possible in case $p \equiv 1 \bmod 2n$ [39].

3.2 Efficient Computation of the NTT

An implementation of the FFT is possible with complexity $\mathcal{O}(n \log n)$. The fast algorithms exploit symmetry introduced by the n-th root of unity ω and can be described recursive, iterative or as a composition of varying size NTTs [10]. In Algorithm 1 [16,10] we just present a common Cooley and Turkey radix-2 decimation in time approach which we have also used for our hardware implementation. This algorithm contains at its core (line 12 and 13) the "butterfly operation" which is the multiplication of the factor $\omega^{N \bmod n}$ with d and the adding or subtraction of the result from c. The *Bit-Reverse(g)* operation is used to reorder the input vector g where the new position of an element at position k is determined by the value obtained by reversing the binary representation of k.

Algorithm 1.1: ITERATIVE-NTT

Input: $g \in \mathbb{Z}_p[\mathbf{x}]$ of degree less than $n = 2^k$ with $k \in \mathbb{N}$ and a primitive n-th root of unity $\omega \in \mathbb{Z}_p$
Output: $y = \mathrm{NTT}_\omega(a)$

```
A ← Bit-Reverse(g)
m ← 2
while m ≤ N do
    s ← 0
    while s < N do
        for i to m/2 − 1 do
            N ← i · n/m
            a ← s + i
            b ← s + i + m/2
            c ← A[a]
            d ← A[b]
            A[a] ← c + ω^(N mod n) d  mod p
            A[b] ← c − ω^(N mod n) d  mod p
        s ← s + m
    m ← m · 2
return A
```

4 Designing an Efficient NTT Polynomial Multiplier on Reconfigurable Hardware

In this section we deal with the overall design of a polynomial multiplier for lattice-based cryptography. For this purpose we also consider previous implementation of the FFT for signal processing and introduce concepts that can also be applied for polynomial multiplication.

4.1 Existing FFT Implementations

The implementation of the FFT in hardware has been considered for more than forty years and a large variety of approaches exist for certain demands regarding throughput, power usage as well as memory and resource consumption or latency [8,44]. One of the two most common design approaches are pipelined implementations [19] where usually one stage of the FFT is performed in one clock cycle in parallel. However, while being extremely fast, this approach is expensive in terms of hardware resources - especially memory and register space. When dedicated memory blocks are available, like on an FPGA, the FFT can also be implemented memory-based [58] with the coefficients and constants being stored in this memory. The processing is then usually performed iteratively by only one or a low number of processing elements (PE) controlled by a digital address generator (DAG). For radix-2 algorithms the PE usually resembles

the well-known butterfly structure and is used to multiply elements of the processed vector by powers of ω (also known as twiddle factors). In some cases the length of the processed sequence is fixed but there exist also very flexible variable length memory-based FFT implementations [23]. A more detailed characterization of FFT architectures can be found in [55]. An implementation of the NTT on graphic cards is described in [20]. A cryptographic processor for performing elliptic curve cryptography (ECC) in the frequency domain is presented in [6].

4.2 Design Decisions for Lattice-Based Cryptography

As lattice-based cryptography is still a very active field of research and the hardness of ideal lattice-based cryptography not fully understood, no "standard" parameters like specific reduction primes or certain numbers of polynomial coefficients have emerged yet. This is different for established cryptosystems, e.g., elliptic curve cryptography (ECC). For the ECC example NIST primes have been specified for efficient modular reduction [28] and just a few parameter sets have to be considered which allows much more optimized implementations. As such standardization is not yet achieved for lattice based cryptography, we have made our implementation of polynomial multiplication as generic as possible. The only general requirement, that can be considered "standard" in the literature is that $p \bmod 2n \equiv 1$, n is a power of two and p is a prime so that the NTT can be efficiently computed in $\mathbb{Z}_p$.

The NTT is not considered as a very efficient method for most applications as approximate solutions in signal processing are usually sufficient (see [45] for calculations of the required accuracy for polynomial multiplication). Especially, the complexity of the butterfly structure in the PE takes a lot of resources as multiplication by the twiddle factor is just a general purpose multiplication followed by a modular reduction. When using the Fermat [1] or Mersenne [47] number theoretic transform (FNT/MNT) the butterfly can be implemented by shifters (as ω can be 2), no ROM for twiddle factors is needed and the modular reduction is also heavily simplified [10, Chap. 10] [7]. However, in this case the transform length has to be doubled (Theorem 1), more storage space for coefficients is needed and the modular reduction has to be performed separately. But the most important observation when reusing the PE for polynomial multiplication is that the component-wise multiplication step $\mathbf{a} \circ \mathbf{b}$ requires a general purpose multiplication and cannot be implemented just with shifters. As a consequence, we decided to reuse the multiplication hardware that we need for the component-wise multiplication step also in the NTT. This makes sense, as such hardware will be instantiated and would idle most of the time which is a waste of resources.

When utilizing (2) of Theorem 2 it seems that more arithmetic operations and additional ROM storage space is necessary due to the needed table entries for ψ and ψ^{-1}. However, compared to a general purpose NTT based on Theorem 1 with a subsequent polynomial modular reduction we are still saving memory resources. When multiplying two polynomial of length n by appending n zeros, we have to store $2n + 2n = 4n$ twiddle factors with $2n$ for the forward and $2n$

for the backward transform. When directly implementing the negative wrapped convolution we can use a transform size of n. In this case we need additional storage for the ψ^i values. But as $\psi^{2^i} = \omega^i$, every second value in the lower half of the ω^i table is also an entry for ψ^i. As a consequence, we are just requiring $2n + 0.5n + 0.5n = 3n$ entries in the ROM and two RAMs, each with n entries when implementing the negative wrapped convolution directly.

5 Implementation Details

In this section we describe our FPGA implementation of a flexible NTT-based polynomial multiplication core specifically designed for high performance lattice-based cryptography.

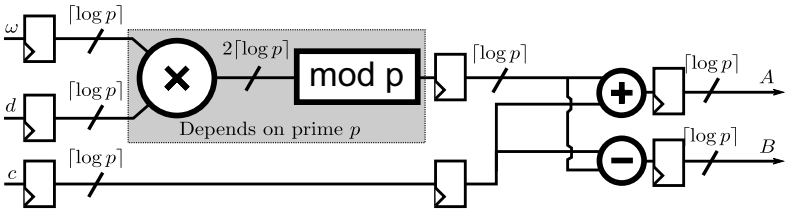

Fig. 1. Block structure of the implemented NTT butterfly which is reused as general purpose multiplier in the polynomial multiplication core. The final addition/subtraction is modulo p.

The processing element (PE) of the polynomial multiplier (depicted in Figure 1) and NTT resembles the common butterfly structure and computes one node of the NTT flow every clock cycle. In order to allow generic usage, a multiplier and a subsequent modular reduction circuit is synthesized for a given reduction prime p. However, it is also possible to write plug-ins that exploit specific structures in the prime number for a more efficient reduction (e.g., for the Fermat prime $2^{16} + 1$) or multiplication (e.g., table lookup for small values of $ab \bmod p$). At synthesis time, it is checked if a specific circuit is available which then takes precedence over the generic implementation. This also allows resource trade-offs, as the plug-ins can be designed to make use of slices, DSPs or table lookups in block memory. The generic multipliers are just instantiations of the Xilinx provided multiplication IP cores for various bit widths (maximum 64×64-bit). The usage of these cores also allows trade-offs between DSPs and logic slices as well as between performance and latency.

The polynomial multiplication unit depicted in Figure 2 uses the memory elements `RAM_A` and `RAM_B`, each of size $n\lceil \log p \rceil$ bits to store the coefficients of the polynomials $\mathbf{a}$ and $\mathbf{b}$. When coefficients a_i or b_i with $0 \leq i < n$ are loaded sequentially into the multiplier, the `BITREV` component ensures that they are multiplied by ψ^i and stored at bit-reversed locations into the RAM. For

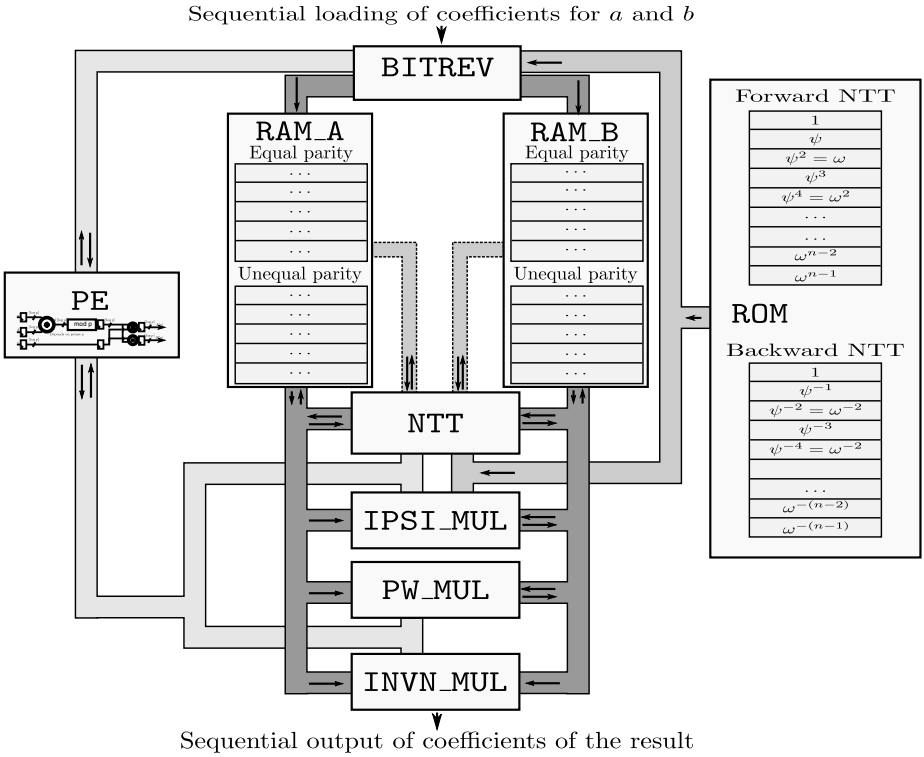

Fig. 2. Architecture of the polynomial multiplication unit. The arrows indicate main data flows on the bus.

this, the modular arithmetic of the shared processing element `PE` is used whose butterfly structure also allows simple addition or multiplication of two values in $\mathbb{Z}_p$. The `ROM` component stores the twiddle factors ω^i as well as ψ^i for $0 \leq i < n$ and requires $3n\lceil \log p \rceil$ bits of block RAM. The `NTT` unit then performs the number theoretic transform in the forward direction in-place on the coefficients in `RAM_A` and `RAM_B`. Then the content is multiplied component-wise by the `PW_Mul` unit with the result being stored in `BRAM_B`. After an inverse NTT (forward NTT with different values from the `ROM` component) the `IPSI_MUL` and `INVN_MUL` components just multiply the output of the inverse NTT by the inverse of n and the inverse values of powers of ψ^i and output the result sequentially.

One of our goals is maximum utilization of the most expensive arithmetic in the `PE` component. Therefore, the `PE` component is shared between all other parts of the design. This yields a near one-hundred percent utilization of the multiplier. Note also, that it would be possible to execute the two forward NTTs in parallel with two `PE`s and `NTT` components. However, during the inverse NTT one of these units would idle which seems not efficient in terms of resource usage.

Integrated block RAM on FPGAs usually supports dual-port access so that only two values can be written and/or read independently in one clock cycle. This is not optimal, as the butterfly structure depicted in Figure 1 needs two inputs

and produces two outputs during one clock cycle. An option would be to just perform one butterfly calculation every two clock cycles but this would double the runtime of the NTT component. Therefore, we use a transparent mechanism for the realization of a conflict free access scheme. We rely on an observation by Pease [44] that the parity of the address lines requesting the inputs and outputs for the butterfly always differ. This division is transparently realized in a wrapper in RAM_A and RAM_B. In Figure 2 this division is also visible as each RAM element contains actually two RAMs (one for equal and another one for unequal parity addresses). This allows standard dual-port access for modules like PW_Mul and also provides two additional pseudo-ports exclusively to the NTT component.

The interface of our implementation consists of a port for coefficients of $\mathbf{a}$ and $\mathbf{b}$ and the result $\mathbf{c}$ is then outputted on another port. Configuration of the core just requires the specification of $n, p, \psi, \psi^{-1}, n^{-1}$. All needed tables in the ROM component are initialized by the synthesizer and mapped into an optimal block RAM structure. Obtaining these values is possible by using brute-force or more complex algorithms that find n-th roots of unity (e.g., with a computer algebra system) and its square root. Our design could even be adapted to other rings or used for general purpose polynomial multiplication but may then not be as competitive as for the $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$ case.

6 Results and Comparison

All results were obtained post place-and-route (PAR), generated with Xilinx ISE 13.3 and implemented stand-alone on the medium-cost Spartan-6 LX100 (speed-grade -3) with some optimizations turned on (using the SmartExplorer [60]). The Spartan-6 LX100 has 180 integrated DSPs, 268 18K block RAMs and supports up to 126K registers and 63K LUTs on 15K slices.

The general performance of the polynomial multiplication mainly depends on the runtime of the NTT core which uses $\approx \frac{n \log n}{2}$ cycles as the butterfly step executes the inner loop of Algorithm 1 every cycle. Therefore, by executing two forward NTTs, multiplying the polynomials component-wise and then executing an inverse NTT we multiply two polynomials in $\mathbb{Z}_p[\mathbf{x}]/\langle x^n + 1\rangle$ in $\approx 3 \cdot (\frac{n \log n}{2}) + 5.5n$ cycles. The additional $5.5n$ cycles are introduced by loading of coefficients $(2n)$, component-wise multiplication $(1.5n)$ and multiplication of the final result by ψ^{-i} and n^{-1} $(2n)$.

The resource consumption of the polynomial multiplier instantiated on the FPGA is mainly influenced by the resource consumption of the $\lceil \log p \rceil \times \lceil \log p \rceil$-bit multiplier and modular p reduction circuit in the butterfly module. For all reduction primes we have designed specialized reduction circuits that take into account the number and position of ones in the binary representation of the prime (see [18,28]). The resource consumption is generally also influenced by the number of coefficients as for larger values of n the address path becomes larger. For our comparison we just consider a scenario in which two dense polynomials (very few zero coefficients) with coefficients in the range from zero to $p - 1$ are multiplied. With one BRAM we denote a full 18K-bit block RAM which can be split into two independent 9K-bit block RAMs (denoted as 0.5 BRAM).

6.1 Polynomial Multiplication in $\mathbb{Z}_{2^{16}+1}\langle x^n+1\rangle$

In Table 1 we detail the performance and resource consumption for the biggest known Fermat prime $p = 2^{16} + 1 = 65537$ which yields a very efficient reduction algorithm [7] and may therefore be used when parameterizing lattice-based schemes. As previously stated, the amount of required slices slowly increases for larger values of n but the limiting factor seems to be the increasing number of occupied block RAMs. However, note that it is still possible to implement the multiplier even for $n = 4096$ on a low-cost FPGA like the Spartan-6 LX16.

Table 1. Resource consumption and performance results (post PAR) of the NTT polynomial multiplier on the Spartan-6 LX100 for the biggest Fermat prime $p = 2^{16} + 1 = 65537$ and a variable number of coefficients n. The column "Mul/s" contains the number of polynomial multiplication that the core can execute per second.

n	p	LUT	FF	Slice	DSP	BRAM	MHz	Cycles	Mul/s
128	65537	1407	1123	535	1	2.5	209	2342	89239
256	65537	1438	1123	520	1	2.5	209	4774	43778
512	65537	1585	1205	615	1	4	196	10014	19572
1024	65537	1644	1241	659	1	6.5	200	21278	9399
2048	65537	1757	1350	707	1	12.5	204	45326	4500
4096	65537	1844	1390	684	1	25	197	96526	2040

6.2 Ring-LWE Encryption

Secure parameters for the multi-bit ring-LWE encryption system are derived in [50]. Arithmetic is defined over the ring $R = \mathbb{Z}_p[\mathbf{x}]/\langle x^n+1\rangle$ and the R-module R^m with elements $\hat{x}$. The first multiplication defined in R^m is just component-wise while the second is a convolution $\circledast : R^m \times R^m \rightarrow R, (\hat{x}, \hat{y}) \mapsto \sum_{i=1}^{m} x_i y_i$. The ciphertext (u, c) is then computed from the public key $pk = (\hat{a} \in R^m, \hat{p} \in R^m)$

$$u = \hat{a} \circledast \hat{r} \in R, c = \hat{p} \circledast \hat{r} + k\frac{p-1}{2} \in R \tag{3}$$

where $\hat{r}$ consists only of m small polynomials with coefficients $(-2, \ldots, 2)$ and the plaintext k only has zero or ± 1 coefficients. The two parameter sets that should offer long term security are $(n = 256, p = 1049089, m = 14)$ and $(n = 512, p = 5941249, m = 16)$. The results in Table 2 show that the multiplications of $a_i r_i$ and $p_i r_i$ can be performed very efficiently with our implementation just requiring $22\mu s$ and $53\mu s$ per polynomial multiplication, respectively.

6.3 Homomorphic Encryption

Our hardware implementation could significantly accelerate homomorphic encryption and is beneficial in terms of energy and device costs. In [43] the implementation and parameterization of a "somewhat" homomorphic encryption

scheme (SHE) in the computer algebra system MAGMA is presented. It is based on ideal lattices with the ring-LWE problem as hardness assumption [40] and allows the computation of a fixed amount of additions and multiplications on encrypted data (depending on the parameters). Encryption requires two multiplications in $\mathbb{Z}_p[\mathbf{x}]/\langle x^n+1\rangle$ which make up for most of the runtime. The number of coefficients of the used polynomials ranges from $n = 512$ to $n = 131072$. A reasonable set is ($n = 1024$,$p = 1061093377$) for the homomorphic computation of the mean and ($n = 2048$, $q = 2^{57} + 25 \cdot 2^{13} + 1 = 144115188076060673$) for the computation of the variance on encrypted data. In Table 2 the performance and resource consumption for these parameter sets is detailed. With a clock frequency of 161 MHz we can compute 3542 polynomial multiplication in one second (0.28 ms per multiplication) for the larger parameter set. In [43] it is reported that the same operation takes 11 ms on a 2.1 GHz dual core processor and thus we achieve a speed-up by a factor of 39 for this operation that contributes heavily to the overall runtime.

Table 2. Resource consumption and performance results for the application of the NTT polynomial multiplier for LWE encryption and SHE. Numbers are post-PAR obtained on a Spartan-6 LX100.

App.	n	p	LUT	FF	Slice	DSP	BRAM	MHz	Cycles	Mul/s
LWE [50]	256	1049089	1637	1507	640	4	5.5	218	4806	45359
LWE [50]	512	5941249	3228	2263	1145	4	7	193	10174	18969
SHE [43]	1024	1061093377	2690	2317	997	4	11.5	194	21405	9063
SHE [43]	2048	$2^{57}+25{\cdot}2^{13}+1$	3846	3986	1310	16	22.5	161	45453	3542

6.4 Comparison with Schoolbook Multiplication

In order to compare our NTT-based results with the naive approach we have developed a basic schoolbook polynomial multiplier according to the description in Section 3. For easier comparison of the resource consumption we reuse the arithmetic already implemented in the NTT butterfly (see Section 5 and Figure 1). Our schoolbook implementation uses $n^2 + 2n$ cycles with the constant $2n$ being attributed to loading of the two inputs while outputting of the result is performed directly in the final round. The multiplier needs three distinct memories (each with n entries) for the storage of the input polynomials a and b as well as for the temporary result. The implementation results for selected parameters are given in Table 3.

We would like to note that the schoolbook approach is generally more versatile than the NTT core (e.g., no restrictions on the reduction prime p) and allows the reduction of runtime or resources when special constrains apply to the input polynomial. The first is the density of the polynomial as a more advanced schoolbook polynomial multiplication circuit (than the one we used) could detect zeros in both polynomials and may avoid the inner multiplications with them. Therefore, for a sparse polynomial the runtime can be calculated as the number of non-zero coefficients in the first polynomial times the number of non-zero

coefficients in the second polynomial. Considering this, our results for the NTT allow an estimation when a schoolbook implementation with detection of sparse coefficients may be advantageous. The resource consumption of the multiplier can be lowered with a special schoolbook multiplier when the coefficients of the two input polynomials are bounded (e.g., only zero or one). This also applies to the polynomial multiplication performed in the ring-LWE encryption scheme but as noted before, only impacts the resource consumption and not the general runtime as a smaller internal multiplier can be used (e.g., 2×21-bit multiplier instead of a 21×21-bit multiplier). This also enables a smaller modular reduction circuit and in some special cases like when one polynomial contains only zero or ones, the multiplier becomes just an adder.

Table 3. Performance of selected instantiations of a schoolbook polynomial multiplier for the multiplication of two dense and equally sized polynomials. Numbers are post PAR obtained on a Spartan-6 LX100.

App.	n	p	LUT	FF	Slice	DSP	BRAM	MHz	Cycles	Mul/s
Fermat	128	65537	547	555	201	1	1.5	320	16670	19196
Fermat	256	65537	495	550	213	1	1.5	300	66078	4540
Fermat	512	65537	507	563	217	1	1.5	302	263198	1147
Fermat	1024	65537	529	576	224	1	3	295	1050654	280
Fermat	2048	65537	679	663	291	1	6	295	4198430	70
Fermat	4096	65537	703	678	304	1	13.5	273	16785438	16
LWE [50]	256	1049089	997	1327	375	4	3	263	66088	3979
SHE [43]	1024	1061093377	1725	2365	687	4	6	185	1050674	147

6.5 Comparison with Related Work

As mentioned in Section 2 some implementations of lattice-based cryptography have been recently proposed. In this section we compare our implementation of polynomial multiplication with relevant implementations on reconfigurable hardware and highlight the different approaches.

For their implementation of the SWIFFT(X) hash function which uses the FFT/NTT as a core primitive Györfi et al. [29] utilize the fact that the small reduction prime $p = 257$ is fixed. This allows the usage of look-up tables instead of DSP-based arithmetic. Furthermore, they utilize the diminished-one number system in order to represent numbers modulo $p = 2^k + 1$. The resource consumption is quite costly with 3,639 slices and 68 BRAMs of a large Virtex-5 LX110T FPGA but their pipelined implementation is able to compute the FFT in one clock cycle per sample with a latency of $\log(n)$ at a frequency of 150 MHz. Note that a fair comparison to our results is not possible as the parameters of SWIFFT ($n = 64, p = 257$) are much smaller parameter than those we have considered.

A complete and fully functional reconfigurable hardware implementation of LWE encryption is given in [26] with a special emphasis on efficient Gaussian

sampling and the parallel implementation of the FFT/NTT. The result is available in $\mathcal{O}(\log n)$ time instead of $\mathcal{O}(n \log n)$ as for our implementation. However, due to this parallel approach the resource consumption of the implementation is excessively high requiring a very large Virtex-7 2000T FPGA at the cost of several thousand US$ per device. For the smallest secure parameter set with $n = 256$ and $p = 7681$ the encryption engine requires 140K registers and 320K LUTs while decryption takes 65K registers and 124K LUTs. The encryption and decryption throughput for this parameter set is approx. 16 MBits/s.

6.6 Optimizations

The given performance figures show that the NTT is fast and resource efficient even on low-cost devices. However, optimizations and extensions of the current architecture are still possible. Recall, that the NTT is performed in place and that the result of the component-wise multiplication is stored in `BRAM_B`. Thus the transformed coefficients in `RAM_A` are only overwritten when a new polynomial multiplication is started. This allows us to keep the transformed value in `BRAM_A` and just read in new coefficients into `BRAM_B`. As a consequence we can flexibly declare one polynomial that has been transformed once as constant and save one forward NTT for further multiplication with this constant.

We also improved the resource consumption by the instantiation of an array of s polynomial multipliers that share the constants stored in the `ROM` component (as depicted in Figure 3 for $s = 3$). As only a global valid signal exists, all multipliers are operated and loaded simultaneously so that the same powers of ψ and ω are needed in every unit. Thus the required amount of memory entries (ROM and RAM) is $s \cdot 2n + 3n \approx s \cdot 2n$ for large values of s. The approximate resource consumption for a multiplier array that simultaneously computes all 28 polynomial ($2m$) multiplications in the two convolutions of the multi-bit ring-LWE encryption function ($n = 256, p = 1049089$) on a Spartan-6 LX100 is 48k slice registers, 39k LUTs, 14k slices, 113 BRAMs and 112 DPSs with a clock frequency of 127 MHz. This allows an impressive amount of 26425 calls to the encryption function per second (omitting the other cheaper or at least parallelizable operations like final summation or error sampling). Combining the array structure with the ability to multiply efficiently by a constant is particularly useful for the ring-LWE encryption system when a large number of encryptions have to be performed for a fixed public key so that $\hat{a}, \hat{p}$ can be treated as transformed quasi-constant and stored in each part of the array.

7 Conclusion and Future Work

In this work we have shown that the underlying arithmetic of lattice-based schemes can be efficiently implemented on FPGAs with a runtime under one millisecond for a great variety of parameters ($n = 128, \ldots, 4096 / \lceil \log p \rceil = 16, \ldots, 58$) that have been previously proposed by other researchers. Our implementation can act as a building block for future implementations and should be helpful in

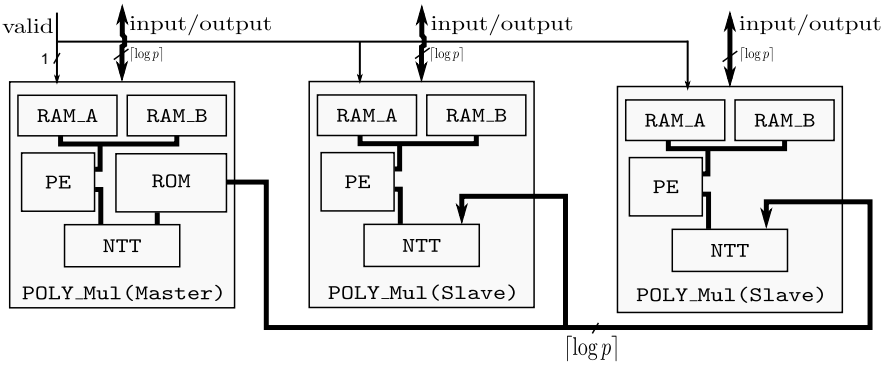

Fig. 3. A master-slave architecture for multiple synchronized polynomial multipliers `Poly_Mul`. Coefficients are simultaneously loaded and outputted as only one global valid signal exists.

order to compare the efficiency of current or future lattice-based proposals with other classical or post-quantum secure alternatives.

In future work we plan to integrate the presented polynomial multiplier into a fully functional implementation of a ring-LWE or homomorphic encryption scheme. Moreover, our implementation should act as the basis for a lattice co-processor with a configurable amount of registers and more versatile operations like addition of constants, adding of two polynomials or random sampling. It may even be possible to define the required operations in a higher level language and to synthesize the processor based on such an abstract description. We would also like to investigate how our direct approach performs in comparison with implementations of the Mersenne or Fermant number transform [7,6] in terms of performance and resource usage. As explained, some corner cases exist in which schoolbook multiplication may be faster or at least more resource efficient than our proposal. We are therefore planning to provide optimized implementations for multiplication of sparse polynomials or polynomials with different coefficient bit-widths. Another interesting topic may be the leakage resilience of the designed multiplier.

References

1. Agarwal, R., Burrus, C.: Fast convolution using fermat number transforms with applications to digital filtering. IEEE Transactions on Acoustics, Speech and Signal Processing 22(2), 87–97 (1974)
2. Ajtai, M.: Generating hard instances of lattice problems. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 99–108. ACM (1996)

3. Arbitman, Y., Dogon, G., Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFTX: A proposal for the SHA-3 standard. Submission to NIST (2008)
4. Atici, A.C., Batina, L., Fan, J., Verbauwhede, I., Yalcin, S.B.O.: Low-cost implementations of NTRU for pervasive security. In: International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2008, pp. 79–84. IEEE (2008)
5. Bailey, D.V., Coffin, D., Elbirt, A., Silverman, J.H., Woodbury, A.D.: NTRU in Constrained Devices. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 262–272. Springer, Heidelberg (2001)
6. Baktir, S., Kumar, S., Paar, C., Sunar, B.: A state-of-the-art elliptic curve cryptographic processor operating in the frequency domain. Mob. Netw. Appl. 12(4), 259–270 (2007)
7. Baktir, S., Sunar, B.: Achieving efficient polynomial multiplication in fermat fields using the fast fourier transform. In: Proceedings of the 44th Annual Southeast Regional Conference, ACM-SE 44, pp. 549–554. ACM, New York (2006)
8. Bergland, G.: Fast fourier transform hardware implementations–an overview. IEEE Transactions on Audio and Electroacoustics 17(2), 104–108 (1969)
9. Bernstein, D.J.: Fast multiplication and its applications. Algorithmic Number Theory 44, 325–384 (2008)
10. Blahut, R.E.: Fast Algorithms for Signal Processing. Cambridge University Press (2010)
11. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 18, p. 111 (2011)
12. Buchmann, J., May, A., Vollmer, U.: Perspectives for cryptographic long-term security. Communications of the ACM 49(9), 50–55 (2006)
13. Buchmann, J., Lindner, R.: Secure Parameters for SWIFFT. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT 2009. LNCS, vol. 5922, pp. 1–17. Springer, Heidelberg (2009)
14. Cheng, L.S., Miri, A., Yeap, T.H.: Efficient FPGA implementation of FFT based multipliers. In: Canadian Conference on Electrical and Computer Engineering, pp. 1300–1303. IEEE (2005)
15. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Math. Comput 19(90), 297–301 (1965)
16. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (July 2009)
17. Corona, C.C., Moreno, E.F., Henriquez, F.R., et al.: Hardware design of a 256-bit prime field multiplier suitable for computing bilinear pairings. In: 2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 229–234. IEEE (2011)
18. Deschamps, J.P., Sutter, G.: Comparison of FPGA implementation of the mod M reduction. Latin American Applied Research 37(1), 93–97 (2007)
19. Dreschmann, M., Meyer, J., Huebner, M., Schmogrow, R., Hillerkuss, D., Becker, J., Leuthold, J., Freude, W.: Implementation of an Ultra-High Speed 256-Point FFT for Xilinx Virtex-6 Devices. In: 2011 9th IEEE International Conference on Industrial Informatics (INDIN), pp. 829–834 (July 2011)
20. Emeliyanenko, P.: Efficient Multiplication of Polynomials on Graphics Hardware. In: Dou, Y., Gruber, R., Joller, J.M. (eds.) APPT 2009. LNCS, vol. 5737, pp. 134–149. Springer, Heidelberg (2009)

21. Frederiksen, T.K.: A practical implementation of Regev's LWE-based cryptosystem (2010), http://daimi.au.dk/~jot2re/lwe/resources/A%20Practical%20Implementation%20of%20Regevs%20LWE-based%20Cryptosystem.pdf
22. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology, pp. 31–51. Springer (2008)
23. Gautam, V., Ray, K.C., Haddow, P.: Hardware efficient design of variable length FFT processor. In: 2011 IEEE 14th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pp. 309–312 (April 2011)
24. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 169–178. ACM (2009)
25. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. IACR Cryptology ePrint Archive, 2012:99 (2012)
26. Göttert, N., Feller, T., Schneider, M., Huss, S.A., Buchmann, J.: On the design of hardware building blocks for modern lattice-based encryption schemes. In: Cryptographic Hardware and Embedded Systems–CHES 2012 (2012)
27. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: A signature scheme for embedded systems. In: Cryptographic Hardware and Embedded Systems–CHES 2012 (2012)
28. Güneysu, T., Paar, C.: Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 62–78. Springer, Heidelberg (2008)
29. Györfi, T., Cret, O., Hanrot, G., Brisebarre, N.: High-throughput hardware architecture for the SWIFFT / SWIFFTX hash functions. In: IACR Cryptology ePrint Archive, 2012:343 (2012)
30. Hoffstein, J., Pipher, J., Silverman, J.: NTRU: A ring-based public key cryptosystem. Algorithmic Number Theory, 267–288 (1998)
31. Kamal, A.A., Youssef, A.M.: An FPGA implementation of the NTRUEncrypt cryptosystem. In: 2009 International Conference on Microelectronics (ICM), pp. 209–212. IEEE (2009)
32. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. Soviet Physics Doklady 7, 595 (1963)
33. 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)
34. Lyubashevsky, V.: Lattice-Based Identification Schemes Secure Under Active Attacks. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 162–179. Springer, Heidelberg (2008)
35. Lyubashevsky, V., Micciancio, D.: Generalized Compact Knapsacks Are Collision Resistant. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part II. LNCS, vol. 4052, pp. 144–155. Springer, Heidelberg (2006)
36. 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)
37. Lyubashevsky, V.: Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (2009)
38. Lyubashevsky, V.: Lattice Signatures without Trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012)

39. 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)
40. 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)
41. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. Computational Complexity 16(4), 365–411 (2007)
42. Micciancio, D., Regev, O.: Lattice-based cryptography. In: Post-Quantum Cryptography, pp. 147–191 (2009)
43. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW 2011, pp. 113–124. ACM, New York (2011)
44. Pease, M.C.: An adaptation of the fast fourier transform for parallel processing. J. ACM 15(2), 252–264 (1968)
45. Percival, C.: Rapid multiplication modulo the sum and difference of highly composite numbers. Mathematics of Computation 72(241), 387–396 (2003)
46. Pollard, J.M.: The fast fourier transform in a finite field. Mathematics of Computation 25(114), 365–374 (1971)
47. Rader, C.M.: Discrete convolutions via mersenne transforms. IEEE Transactions on Computers 100(12), 1269–1273 (1972)
48. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC 2005: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, USA, May 22-24, p. 84. ACM Press (2005)
49. Regev, O.: The learning with errors problem. Invited Survey in CCC (2010)
50. Rückert, M., Schneider, M.: Estimating the security of lattice-based cryptosystems. Cryptology ePrint Archive, Report 2010/137 (2010), http://eprint.iacr.org/
51. Schönhage, A., Strassen, V.: Schnelle Multiplikation Grosser Zahlen. Computing 7(3), 281–292 (1971)
52. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 1994 Proceedings of 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE Computer Society Press, Los Alamitos (1994)
53. Shoup, V.: NTL: A library for doing number theory (2001)
54. Stehlé, D., Steinfeld, R.: Making NTRU as Secure as Worst-Case Problems Over Ideal Lattices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 27–47. Springer, Heidelberg (2011)
55. Suleiman, A., Saleh, H., Hussein, A., Akopian, D.: A family of scalable FFT architectures and an implementation of 1024-point radix-2 FFT for real-time communications. In: IEEE International Conference on Computer Design, ICCD 2008, pp. 321–327 (October 2008)
56. von zur Gathen, J., Shokrollahi, J.: Efficient FPGA-based Karatsuba multipliers for polynomials over $\mathbb{F}_2$. In: Selected Areas in Cryptography, pp. 359–369. Springer (2006)
57. Weimerskirch, A., Paar, C.: Generalizations of the Karatsuba algorithm for polynomial multiplication (2003)
58. Wey, C.-L., Lin, S.-Y., Tang, W.-C.: Efficient memory-based FFT processors for OFDM applications. In: 2007 IEEE International Conference on Electro/Information Technology, pp. 345–350 (May 2007)

59. Winkler, F.: Polynomial Algorithms in Computer Algebra (Texts and Monographs in Symbolic Computation), 1st edn. Springer (August 1996)
60. Xilinx. Smartxplorer for ISE project navigator users, Version 12.1 (2010), http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/ug689.pdf
61. Yao, Y., Huang, J., Khanna, S., Shelat, A., Calhoun, B.H., Lach, J., Evans, D.: A sub-0.5V lattice-based public-key encryption scheme for RFID platforms in 130nm CMOS. In: Workshop on RFID Security (RFIDsec 2011 Asia), Cryptology and Information Security, pp. 96–113. IOS Press (April 2011)

The Security Impact of a New Cryptographic Library

Daniel J. Bernstein[1], Tanja Lange[2], and Peter Schwabe[3]

[1] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7053, USA
djb@cr.yp.to
[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600MB Eindhoven, The Netherlands
tanja@hyperelliptic.org
[3] Research Center for Information Technology Innovation and
Institute of Information Science
Academia Sinica
No. 128 Academia Road, Section 2, Nankang, Taipei 11529, Taiwan
peter@cryptojedi.org

Abstract. This paper introduces a new cryptographic library, NaCl, and explains how the design and implementation of the library avoid various types of cryptographic disasters suffered by previous cryptographic libraries such as OpenSSL. Specifically, this paper analyzes the security impact of the following NaCl features: no data flow from secrets to load addresses; no data flow from secrets to branch conditions; no padding oracles; centralizing randomness; avoiding unnecessary randomness; extremely high speed; and cryptographic primitives chosen conservatively in light of the cryptanalytic literature.

Keywords: confidentiality, integrity, simplicity, speed, security.

1 Introduction

For most cryptographic operations there exist widely accepted standards, such as the Advanced Encryption Standard (AES) for secret-key encryption and 2048-bit RSA for public-key encryption. These primitives have been extensively studied, and breaking them is considered computationally infeasible on any existing computer cluster.

This work was supported by the National Science Foundation under Grant 1018836; by the European Commission through the ICT Programme under Contract ICT-2007-216499 CACE and Contract ICT-2007-216676 ECRYPT II; by the US Air Force Office of Applied Research & Development under Grant AOARD-11-4092; and by the National Science Council, National Taiwan University and Intel Corporation under Grant NSC-100-2911-I-002-001 and 101R7501. Part of this work was carried out when Peter Schwabe was employed by National Taiwan University; part of this work was carried out when Peter Schwabe was employed by Technische Universiteit Eindhoven. Permanent ID of this document: 5f6fc69cc5a319aecba43760c56fab04. Date: 2012.07.31.

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 159–176, 2012.

For each of these cryptographic primitives there exist various implementations and software libraries, and it has become common best practice in the development of secure systems to use the implementations in these libraries as building blocks. One should thus expect that the cryptographic layer of modern information systems does not expose any vulnerabilities to attackers. Unfortunately this expectation is far from reality, as demonstrated by one embarrassing cryptographic failure after another.

A New Cryptographic Library: NaCl. To address the underlying problems we have designed and implemented a new cryptographic library. The library name, NaCl, is pronounced "salt" and stands for "Networking and Cryptography Library". This paper discusses only the cryptographic part of NaCl; the networking part is still in prototype form.

NaCl is in the public domain and is available from `http://nacl.cr.yp.to` and `http://nacl.cace-project.eu`, along with extensive documentation. The signature component of NaCl is integrated only into the latest development version, which is not yet online, but the same code is available separately as part of the SUPERCOP benchmarking package at `http://bench.cr.yp.to`. NaCl steers clear of all patents that we have investigated and has not received any claims of patent infringement.

The first announcement of NaCl was in 2008. We considered changing the name of the project when Google announced Native Client, but decided that there was no real risk of confusion. The first release of NaCl was in 2009 but was missing some of the important features discussed in this paper; the C++ NaCl API was not released until 2010, for example, and signatures were not released until 2011.

A research paper on cryptographic software normally focuses on optimizing the choice and implementation of a single cryptographic primitive at a specified security level: for example, [11] reports speed records for signatures at a 2^{128} security level. This paper is different. Our goal is to analyze the real-world security benefits of switching from an existing cryptographic library such as OpenSSL [30] to a completely new cryptographic library. Some of these security benefits are tied to performance, as discussed later, so we naturally use the results of papers such as [11]; but what is new in this paper is the security analysis.

Credits. Several of the implementations used in NaCl are partially or entirely from third parties. The portability of NaCl relies on the `ref` implementation of Curve25519 written by Matthew Dempsky (Mochi Media, now Google). From 2009 until 2011 the speed of NaCl on common Intel/AMD CPUs relied on the `donna` and `donna_c64` implementations of Curve25519 written by Adam Langley (Google) — which, interestingly, also appear in Apple's acknowledgments [4] for iOS 4. The newest implementations of Curve25519 and Ed25519 were joint work with Niels Duif (Technische Universiteit Eindhoven) and Bo-Yin Yang (Academia Sinica). The `core2` implementation of AES was joint work with Emilia Käsper (Katholieke Universiteit Leuven, now Google).

Prototype Python wrappers around C NaCl have been posted by Langley; by Jan Mojzis; and by Sean Lynch (Facebook). We will merge these wrappers and

integrate them into the main NaCl release as a single supported Python NaCl, in the same way that we support C++ NaCl.

2 Background for the Security Analysis: The NaCl API

The reader is assumed to be familiar with the fact that most Internet communication today is cryptographically unprotected. The primary goal of NaCl is to change this: to cryptographically protect every network connection, providing strong confidentiality, strong integrity, and state-of-the-art availability against attackers sniffing or modifying network packets.

Confidentiality is limited to packet contents, not packet lengths and timings, so users still need anti-traffic-analysis tools: route obfuscators such as Tor [38], timing obfuscators, etc. Of course, users also need vastly better software security in operating systems, web browsers, document viewers, etc. Cryptography is only one part of security.

This section introduces the functions provided by NaCl, with an emphasis on the simplicity of these functions: more precisely, the simplicity that these functions bring to cryptographic applications. There are enough differences between the NaCl API and previous APIs to justify a discussion of the details. Subsequent sections of the paper analyze the security benefits of NaCl, starting from the background provided in this section.

The `crypto_box` API. The central job of a cryptographic library is **public-key authenticated encryption**. The general setup is that a sender, Alice, has a packet to send to a receiver, Bob. Alice scrambles the packet using Bob's public key and her own secret key. Bob unscrambles the packet using Alice's public key and his own secret key. "Encryption" refers to confidentiality: an attacker monitoring the network is unable to understand the scrambled packet. "Authenticated" refers to integrity: an attacker modifying network packets is unable to change the packet produced by Bob's unscrambling. (Availability, to the extent that it is not inherently limited by network resources, is provided by higher-level networking protocols that retransmit lost packets.)

A typical cryptographic library uses several steps to authenticate and encrypt a packet. Consider, for example, the following typical combination of RSA, AES, etc.:

- Alice generates a random AES key.
- Alice uses the AES key to encrypt the packet.
- Alice hashes the encrypted packet using SHA-256.
- Alice reads her RSA secret key from "wire format."
- Alice uses her RSA secret key to sign the hash.
- Alice reads Bob's RSA public key from wire format.
- Alice uses Bob's public key to encrypt the AES key, hash, and signature.
- Alice converts the encrypted key, hash, and signature to wire format.
- Alice concatenates with the encrypted packet.

Often even more steps are required for storage allocation, error handling, etc.

NaCl gives Alice a simple high-level `crypto_box` function that does everything in one step, putting a packet into a box that is protected against espionage and sabotage:

```
c = crypto_box(m,n,pk,sk)
```

The function takes the sender's secret key `sk` (32 bytes), the recipient's public key `pk` (also 32 bytes), a packet `m`, and a nonce `n` (24 bytes), and produces an authenticated ciphertext `c` (16 bytes longer than `m`). All of these objects are C++ `std::string` variables, represented in wire format as sequences of bytes suitable for transmission; the `crypto_box` function automatically handles all necessary conversions, initializations, etc. Bob's operation is just as easy, with the keys and packets reversed, using his secret key, Alice's public key, and the same nonce:

```
m = crypto_box_open(c,n,pk,sk)
```

Each side begins with

```
pk = crypto_box_keypair(&sk)
```

to generate a secret key and a public key in the first place.

These C++ functions are wrappers around C functions; the C functions can also be used directly by C applications. The C NaCl API has the same function names but more arguments: for example, `std::string m` is replaced by `unsigned char *m` and `unsigned long long mlen`, and `std::string c` is replaced by `unsigned char *c`. The formats of `m` and `c` in the C NaCl API are padded so that `clen` matches `mlen`, removing the need to pass `clen` explicitly and allowing ciphertexts to be stored on top of plaintexts. Failures are indicated by exceptions in C++ NaCl and a `-1` return value in C NaCl.

Validation of the API. The API described above might seem *too* simple to support the needs of real-world applications. We emphasize that NaCl has already been integrated into high-security applications that are running on the Internet today.

DNSCurve [9], designed by the first author, provides high-security authenticated encryption for Domain Name System (DNS) queries between a DNS resolver and a DNS server. (The server's public key is provided by its parent DNS server, which of course also needs to be secured; the client's public key is provided as part of the protocol.) NaCl has been used successfully for several independent DNSCurve implementations, including an implementation used [19] by the OpenDNS resolvers, which handle billions of DNS queries a day from millions of computers and automatically use DNSCurve for any DNSCurve server. OpenDNS has also designed and deployed DNSCrypt, a variant of DNSCurve that uses NaCl to authenticate and encrypt DNS queries from a DNS *client* to a DNS resolver; two months after the introduction of DNSCrypt, [40] stated that DNSCrypt was already in use by tens of thousands of clients. Other applications of NaCl so far include the QuickTun VPN software [33]; the Ethos operating

system [35]; and the first author's prototype implementation of CurveCP [10], a high-security cryptographic version of TCP.

C NaCl allows `crypto_box` to be split into two steps, `crypto_box_beforenm` followed by `crypto_box_afternm`, slightly compromising simplicity but gaining extra speed as discussed in Section 4. The `beforenm` step preprocesses `pk` and `sk`, preparing to handle any number of messages; the `afternm` step handles `n` and `m`. Most applications actually use this two-step procedure.

Nonces. The `crypto_box` API leaves nonce generation to the caller. This is not meant to suggest that nonce generation is not part of the cryptographer's job; on the contrary, we believe that cryptographers should take responsibility not just for nonces but also for other security aspects of high-level network protocols. The exposure of nonces simply reflects the fact that nonces are integrated into high-level protocols in different ways.

It might seem simplest to always generate a random 24-byte nonce `n`, and to transmit this nonce as part of the authenticated ciphertext; 24-byte random strings have negligible chance of colliding. If ciphertexts are long then one can tolerate the costs of generating this randomness and of expanding each ciphertext by 24 bytes. However, random nonces do nothing to stop the simplest type of forgery, namely a replay. One standard strategy to prevent replays is to include an increasing number in each packet and to reject any packet whose number is not larger than the number in the last verified packet; using these sequence numbers as nonces is simpler than giving each packet a number *and* a random nonce. On the other hand, choosing public nonces as sequence numbers means giving away traffic information that would otherwise be somewhat more expensive for an attacker to collect. Several different solutions appear in the literature; constraints on nonce generation are often tied directly to questions of the security and privacy that users expect.

Current applications of NaCl, such as DNSCurve and CurveCP, have different requirements regarding nonces, replays, forward secrecy, and many other security issues at a higher level than the `crypto_box` API. A nonceless API would require higher-level complications in all of these applications, and would not simplify their security analysis.

The `crypto_sign` API. Sometimes confidentiality is irrelevant: Alice is sending a public message to many people. In this situation it is helpful for a cryptographic library to provide **public-key signatures**: Alice scrambles the message using her own secret key, and Bob unscrambles the message using Alice's public key. Alice's operations are independent of Bob, allowing the scrambled message to be broadcast to any number of receivers. Signatures also provide non-repudiation, while authenticators are always repudiable.

NaCl provides simple high-level functions for signatures: Alice uses

```
pk = crypto_sign_keypair(&sk)
```

to generate a key pair (again 32 bytes for the public key but 64 bytes for the secret key), and

```
sm = crypto_sign(m,sk)
```

to create a signed message (64 bytes longer than the original message). Bob uses

```
m = crypto_sign_open(sm,pk)
```

to unscramble the signed message, recovering the original message.

Comparison to Previous Work. NaCl is certainly not the first cryptographic library to promise a simple high-level API. For example, Gutmann's cryptlib library [22] advertises a "high-level interface" that "provides anyone with the ability to add strong security capabilities to an application in as little as half an hour, without needing to know any of the low-level details that make the encryption or authentication work." See [23, page 1].

There are, however, many differences between high-level APIs, as illustrated by the following example. The first code segment in the cryptlib manual [23, page 13] ("the best way to illustrate what cryptlib can do") contains the following six function calls, together with various comments:

```
cryptCreateEnvelope( &cryptEnvelope, cryptUser,
   CRYPT_FORMAT_SMIME );
cryptSetAttributeString( cryptEnvelope,
   CRYPT_ENVINFO_RECIPIENT,
   recipientName, recipientNameLength );
cryptPushData( cryptEnvelope, message, messageSize,
   &bytesIn );
cryptFlushData( cryptEnvelope );
cryptPopData( cryptEnvelope, encryptedMessage, encryptedSize,
   &bytesOut );
cryptDestroyEnvelope( cryptEnvelope );
```

This sequence has a similar effect to NaCl's

```
c = crypto_box(m,n,pk,sk)
```

where `message` is the plaintext `m` and `encryptedMessage` is the ciphertext `c`.

The most obvious difference between these examples is in conciseness: cryptlib has separate functions

- `cryptCreateEnvelope` to allocate storage,
- `cryptSetAttributeString` to specify the recipient,
- `cryptPushData` to start the plaintext input,
- `cryptFlushData` to finish the plaintext input,
- `cryptPopData` to extract the ciphertext, and
- `cryptDestroyEnvelope` to free storage,

while NaCl handles everything in one function. The cryptlib program must also call `cryptInit` at some point before this sequence.

A much less obvious difference is in reliability. For example, if the program runs out of memory, NaCl will raise an exception, while the above cryptlib code

will fail in unspecified ways, perhaps silently corrupting or leaking data. The cryptlib manual [23, page 35] states that the programmer is required to check that each function returns `CRYPT_OK`, and that the wrong code shown above is included in the manual "for clarity". Furthermore, [23, page 53] says that if messages are large then "only some of the data may be copied in" by `cryptPushData`; the programmer is required to check `bytesIn` and loop appropriately. Trouble can occur even if messages are short and memory is ample: for example, [23, page 14] indicates that recipient public keys are retrieved from an on-disk database, but does not discuss what happens if the disk fails or if an attacker consumes all available file descriptors.

Some of the differences between these code snippets are really differences between C and C++: specifically, NaCl benefits from C++ exceptions and C++ strings, while cryptlib does not use these C++ features. For applications written in C, rather than C++, the cryptlib API should instead be compared to the C NaCl API:

```
crypto_box(c,m,mlen,n,pk,sk)
```

This C NaCl function cannot raise C++ exceptions, but it also does not need to: its only possible return value is 0, indicating successful authenticated encryption. C NaCl is intended to be usable in operating-system kernels, critical servers, and other environments that cannot guarantee the availability of large amounts of heap storage but that nevertheless rely on their cryptographic computations to continue working. In particular, C NaCl functions do not call `malloc`, `sbrk`, etc. They do use small amounts of stack space; these amounts will eventually be measured by separate benchmarks, so that stack space can be allocated in advance and guaranteed to be adequate.

Perhaps the most important difference between these NaCl and cryptlib examples is that the `crypto_box` output is authenticated and encrypted using keys from Alice and Bob, while the cryptlib output is merely encrypted to Bob without any authentication; cryptlib supports signatures but does not add them without extra programming work. There is a long history of programs omitting cryptographic authentication, incorrectly treating all successfully decrypted data as authentic, and being exploited as a result; with cryptlib, writing such programs is easier than writing programs that include proper authentication. With NaCl, high-security authenticated encryption is the easiest operation.

3 Core Security Features and Their Impact

This section presents various case studies of cryptographic disasters, and explains the features of NaCl that eliminate these types of disasters.

Two specific types of disasters are addressed in subsequent sections: Section 4 discusses users deliberately weakening or disabling cryptography to address cryptographic performance problems; Section 5 discusses cryptographic primitives being broken.

No Data Flow from Secrets to Load Addresses. In 2005, Osvik, Shamir, and Tromer described a timing attack that discovered the AES key of the `dm-crypt` hard-disk encryption in Linux in just 65 milliseconds. See [31] and [39]. The attack process runs on the same machine but does not need any privileges (for example, it can run inside a virtual machine) and does not exploit any kernel software security holes.

This attack is possible because almost all implementations of AES, including the Linux kernel implementation, use fast lookup tables as recommended in the initial AES proposal; see [18, Section 5.2]. The secret AES key inside the kernel influences the table-load addresses, which in turn influence the state of the CPU cache, which in turn influences measurable timings of the attack process; the attack process computes the AES key from this leaked information.

NaCl avoids this type of disaster by systematically avoiding all loads from addresses that depend on secret data. All of the implementations are thus inherently protected against cache-timing attacks. This puts constraints on the implementation strategies used throughout NaCl, and also influences the choice of cryptographic algorithms in NaCl, as discussed in Section 5.

For comparison, Gutmann's cryptlib manual [23, pages 63–64] claims that cache-timing attacks (specifically "observing memory access latencies for cached vs. un-cached data") and branch-timing attacks (see below) provide almost the same "level of access" as "an in-circuit emulator (ICE)" and that there are therefore "no truly effective defences against this level of threat". We disagree. Software side channels on common CPUs include memory addresses and branch conditions but do not include, e.g., the inputs and outputs to a XOR operation; it is well known that the safe operations are adequate in theory to perform cryptographic computations, and NaCl demonstrates that the operations are also adequate in practice. Typical cryptographic code uses unsafe operations, and cache-timing attacks have been repeatedly demonstrated to be effective against such code, but NaCl's approach makes these attacks completely ineffective.

OpenSSL has responded to cache-timing attacks in a different way, not prohibiting secret load addresses but instead using complicated countermeasures intended to obscure the influence of load addresses upon the cache state. This obviously cannot provide the same level of confidence as the NaCl approach: a straightforward code review can convincingly verify the predictability of all load addresses in NaCl, while there is no similarly systematic way to verify the efficacy of other countermeasures. The review of load addresses and branch conditions (see below) can be automated, as explained in [28] and [27], and in fact has already been formalized and automated for large parts of NaCl; see [3] (which comments that "NaCl code follows strict coding policies that make it *formal verification-friendly*" and explains how parts of the code were verified).

No Data Flow from Secrets to Branch Conditions. Brumley and Tuveri announced in 2011 that they had used a remote timing attack to find the ECDSA private key used for server authentication in a TLS handshake. See [15]. The implementation targeted in this attack is the ECDSA implementation in OpenSSL.

The underlying problem is that most scalar-multiplication (and exponentiation) algorithms involve data flow from secret data into branch conditions: i.e., certain operations are carried out if and only if the key has certain properties. In particular, the OpenSSL implementation of ECDSA uses one of these algorithms. Secret data inside OpenSSL influences the state of the CPU branch unit, which in turn influences the amount of time used by OpenSSL, which in turn influences measurable timings of network packets; the attacker computes the ECDSA key from this leaked information.

NaCl avoids this type of disaster by systematically avoiding all branch conditions that depend on secret data. This is analogous to the prohibition on secret load addresses discussed above; it has pervasive effects on NaCl's implementation strategies and interacts with the cryptographic choices discussed in Section 5.

No Padding Oracles. In 1998 Bleichenbacher successfully decrypted an RSA-encrypted SSL ciphertext by sending roughly one million variants of the ciphertext to the server and observing the server's responses. The server would apply RSA decryption to each variant and publicly reject the (many) variants not having "PKCS #1" format. Subsequent integrity checks in SSL would defend against forgeries and reject the remaining variants, but the pattern of initial rejections already leaked so much information that Bleichenbacher was able to compute the plaintext. See [14].

NaCl has several layers of defense against this type of disaster:

- NaCl's authenticated-encryption mechanism is designed as a secure unit, always wrapping encryption inside authentication. Nothing is decrypted unless it first survives authentication, and the authenticator's entire job is to prevent the attacker from forging messages that survive authentication.
- Forged messages always follow the same path through authenticator verification, using constant time (depending only on the message length, which is public) and then rejecting the message, with no output other than the fact that the message is forged.
- Even if the attacker forges a variant of a message by sheer luck, the forgery will be visible only through the receiver accepting the message, and standard nonce-handling mechanisms in higher-level protocols will instantly reject any further messages under the same nonce. NaCl derives new authentication and encryption keys for each nonce, so the attacker will have no opportunity to study the effect of those keys on any further messages.

Note that the third defense imposes a key-agility requirement on the underlying cryptographic algorithms.

Most cryptographic libraries responded to Bleichenbacher's attack by trying to hide different types of message rejection, along the lines of the second defense; for example, [24] shows that this approach was adopted by the GnuTLS library in 2006. However, typical libraries continue to show small timing variations, so this defense by itself is not as confidence-inspiring as using strong authentication to shield decryption. Conceptually similar attacks have continued to plague cryptographic software, as illustrated by the SSH attack in [1] in 2009 and the very recent DTLS attack in [2].

Centralizing Randomness. In 2006 a Debian developer removed one critical line of randomness-generation code from the OpenSSL package shipped with Debian GNU/Linux. Code-verification tools had complained that the line was producing unpredictable results, and the developer did not see why the line was necessary. Until this bug was discovered in 2008 (see [34]), OpenSSL keys generated under Debian and Ubuntu were chosen from a set of size only 32768. Breaking the encryption or authentication of any communication secured with such a key was a matter of seconds.

NaCl avoids this type of disaster by simply reading bytes from the operating-system kernel's cryptographic random-number generator. Of course, the relevant code in the kernel needs to be carefully written, but reviewing that code is a much more tractable task than reviewing all of the separate lines of randomness-generation code in libraries that decide to do the job themselves. The benefits of code minimization are well understood in other areas of security; we are constantly surprised by the amount of unnecessary complexity in cryptographic software.

A structural deficiency in the `/dev/urandom` API provided by Linux, BSD, etc. is that using it can fail, for example because the system has no available file descriptors. In this case NaCl waits and tries again. We recommend that operating systems add a reliable `urandom(x,xlen)` system call.

Avoiding Unnecessary Randomness. Badly generated random numbers were also involved in the recent collapse of the security system of Sony's PlayStation 3 gaming console. Sony used the standard elliptic-curve digital-signature algorithm, ECDSA, but ignored the ECDSA requirement of a new random secret for each message: Sony simply used a constant value for all messages. Attackers exploited this mistake to compute Sony's root signing key, as explained in [16, slides 122–130], breaking the security system of the PlayStation 3 beyond repair.

NaCl avoids this type of disaster by using *deterministic* cryptographic operations to the extent possible. The `keypair` operations use new randomness, but all of the other operations listed above produce outputs determined entirely by their inputs. Of course, this imposes a constraint upon the underlying cryptographic primitives: primitives that use randomness, such as ECDSA, are rejected in favor of primitives that make appropriate use of pseudorandomness.

Determinism also simplifies testing. NaCl includes a battery of automated tests shared with eBACS (ECRYPT Benchmarking of Cryptographic Systems), an online cryptographic speed-measurement site [12] designed by the first two authors; this site has received, and systematically measured, 1099 implementations of various cryptographic primitives from more than 100 people. The test battery found, for example, that software for a cipher named Dragon was sometimes reading outside its authorized input arrays; the same software had passed previous cryptographic test batteries. All of the core NaCl functions have also been tested against pure Python implementations, some written ourselves and some contributed by Matthew Dempsky.

4 Speed and Its Security Impact

Cryptographic performance problems have frequently caused users to reduce their cryptographic security levels or to turn off cryptography entirely. Consider the role of performance in the following examples:

- `https://sourceforge.net/account` is protected by SSL, but `https://sourceforge.net/develop` redirects the user's web browser to `http://sourceforge.net/develop`, actively *turning off SSL* and exposing the web pages to silent modification by sniffing attackers. Cryptography that is not actually used can be viewed as the ultimate disaster, providing no more security than any of the other cryptographic disasters discussed in this paper.
- OpenSSL's AES implementations continue to use table lookups on most CPUs, rather than obviously safe bitsliced computations that would be slower on those CPUs. The table lookups have been augmented with several complicated countermeasures that are hoped to protect against the cache-timing attacks discussed in Section 3.
- Google has begun to allow SSL for more and more services, but only with a 1024-bit RSA key, despite
  - recommendations from the RSA company to move up to at least 2048-bit RSA by the end of 2010;
  - the same recommendations from the U.S. government; and
  - analyses from 2003 concluding that 1024-bit RSA was already breakable in under a year using hardware that governments and large companies could already afford.

  See, e.g., [32] for an analysis by Shamir (the S in RSA) and Tromer; [25] for an end-of-2010 recommendation from the RSA company; and [5] for an end-of-2010 recommendation from the U.S. government.
- DNSSEC recommends, and uses, 1024-bit RSA for practically all signatures rather than 2048-bit RSA, DSA, etc.: "In terms of performance, both RSA and DSA have comparable signature generation speeds, but DSA is much slower for signature verification. Hence, RSA is the recommended algorithm. ... The choice of key size is a tradeoff between the risk of key compromise and performance. ... RSA-SHA1 (RSA-SHA-256) until 2015, 1024 bits." See [17].
- The Tor anonymity network [38] also uses 1024-bit RSA.

Speed of NaCl. We do not provide any low-security options in NaCl. For example, we do not allow encryption without authentication; we do not allow any data flow from secrets to load addresses or branch conditions; and we do not allow cryptographic primitives breakable in substantially fewer than 2^{128} operations, such as RSA-2048.

The remaining risk is that users find NaCl too slow and turn it off, replacing it with low-security cryptographic software or no cryptography at all. NaCl avoids this type of disaster by providing exceptionally high speeds. NaCl is generally much faster than previous cryptographic libraries, even if those libraries are

asked for lower security levels. More to the point, NaCl is fast enough to handle packet rates beyond the worst-case packet rates of a typical Internet connection.

For example, using a single AMD Phenom II X6 1100T CPU (6 cores, 3.3GHz, purchased for $190 a year ago), NaCl performs

- more than 80000 `crypto_box` operations (public-key authenticated encryption) per second;
- more than 80000 `crypto_box_open` operations (public-key authenticator verification and decryption) per second;
- more than 70000 `crypto_sign_open` operations (signature verification) per second; and
- more than 180000 `crypto_sign` operations (signature generation) per second

for any common packet size. To put these numbers in perspective, imagine a connection flooded with 50-byte packets, each requiring a `crypto_box_open`; 80000 such packets per second would consume 32 megabits per second even without packet overhead. A lower volume of network traffic means that the CPU needs only a fraction of its time to handle the cryptography.

NaCl provides even better speeds than this, for four reasons:

- NaCl uses a single public-key operation for a packet of any size, allowing large packets to be handled with very fast secret-key cryptography; 80000 1500-byte packets per second would fill up a gigabit-per-second link.
- A single public-key operation is shared by many packets from the same public key, allowing all the packets to be handled with very fast secret-key cryptography, if the caller splits `crypto_box` into `crypto_box_beforenm` and `crypto_box_afternm`.
- NaCl uses "encrypt-then-MAC", so forged packets are rejected without being decrypted; a flood of forgeries thus has even more trouble consuming CPU time.
- The signature system in NaCl supports fast batch verification, effectively doubling the speed of verifying a stream of valid signatures.

Most of these speedups do not reduce the cost of handling forgeries under *new* public keys, but a flooded server can continue providing very fast service to public keys that are *already known*.

Comparison to Previous Work. See [7], [11], and [12] for detailed surveys of previous speeds. We give an example here of widely used cryptographic software running much more slowly than NaCl; this type of slowness plays an obvious role in the examples at the beginning of the section.

We ran `openssl speed` on the same AMD Phenom II X6 1100T CPU mentioned above. The OpenSSL version shipped with the operating system (64-bit Ubuntu 11.10) was 1.0.0e, released September 2011. OpenSSL reports speeds on just 1 out of the 6 CPU cores, so we multiplied its operation counts by 6, optimistically assuming that separate operations on separate CPU cores would not interfere with each other.

For public-key authenticated encryption at a security level we would accept, the fastest option in OpenSSL was `nistp256` ECDH (plus secret-key cryptography), running at 9300 operations/second for small packets. (The lowest-security option in OpenSSL was `secp160r1` ECDH, running at 29800 operations/second.) For signing at the same security level, the fastest option was `nistp256` ECDSA, running at 37700 operations/second, but with verification running at only 7800 operations/second. RSA-2048 is much faster for encryption and signature verification, running at 102500 operations/second, but much slower for decryption and signing, running at 2800 operations/second.

Embedded Systems. The optimized implementations in the current version of NaCl are aimed at large CPUs, but all of the cryptographic primitives in NaCl can fit onto much smaller CPUs: there are no requirements for large tables or complicated code. NaCl also makes quite efficient use of bandwidth: as mentioned earlier, public keys are only 32 bytes, signed messages are only 64 bytes longer than unsigned messages, and authenticated ciphertexts are only 16 bytes longer than plaintexts.

The first and third authors of this paper reported in [13] new implementations of the NaCl public-key primitives running at more than 1000 operations/second on an ARM Cortex A8 core (e.g., the Apple A4 CPU in the iPad 1 and iPhone 4). We are integrating this software into NaCl.

5 Cryptographic Primitives in NaCl

Stevens, Sotirov, Appelbaum, Lenstra, Molnar, Osvik, and de Weger announced in 2008 (see [36] and [37]) that, by exploiting various weaknesses that had been discovered in the MD5 hash function, they had created a rogue CA certificate. They could, if they wanted, have impersonated any SSL site on the Internet.

This type of disaster, cryptographic primitives being broken, is sometimes claimed to be prevented by cryptographic standardization. However, there are many examples of standards that have been publicly broken, including DES, 512-bit RSA, and these MD5-based certificates. More to the point, there are some existing standards that can reach NaCl's speeds, but those standards fall far short of NaCl's security requirements.

Our main strategy for avoiding dangerous primitives in NaCl has been to *pay attention to cryptanalysis*. There is an extensive cryptanalytic literature exploring the limits of attacks on various types of cryptographic primitives; some cryptographic structures are comfortably beyond these limits, while others inspire far less confidence. This type of security evaluation is only loosely related to standardization, as illustrated by the following example: Dobbertin, Bosselaers, and Preneel wrote "It is anticipated that these techniques can be used to produce collisions for MD5 and perhaps also for RIPEMD" in 1996 [20], eight years before collisions in MD5 (and RIPEMD) were published and a decade before most MD5-based standards were withdrawn. They recommended switching to RIPEMD-160, which fifteen years later has still not been publicly broken.

This strategy, choosing cryptographic algorithms in light of the cryptanalytic literature, has given us higher confidence in NaCl's cryptographic primitives than in most standards. At the same time this strategy has given us the flexibility needed to push NaCl to extremely high speeds, avoiding the types of disasters discussed in Section 4.

The rest of this section discusses the cryptographic primitives used in NaCl, and explains why we expect these choices to reduce the risk of cryptographic disasters. Specifically, NaCl uses elliptic-curve cryptography, not RSA; it uses an elliptic curve, Curve25519, that has several advanced security features; it uses Salsa20, not AES (although it does include an AES implementation on the side); it uses Poly1305, not HMAC; and for an elliptic-curve signature system it uses EdDSA, not ECDSA.

We are aware that many existing protocols require AES and RSA, and that taking advantage of NaCl as described in this paper requires those protocols to be upgraded. We have prioritized security over compatibility, and as a consequence have also prioritized speed over compatibility. There are other projects that have explored the extent to which speed and security can be improved without sacrificing compatibility, but NaCl is aiming at a different point in the design space, and at applications that are not well served by the existing protocols. DNSCrypt (see Section 2) illustrates the feasibility of our deployment approach.

Cryptographic Choices in NaCl. RSA is somewhat older than elliptic-curve cryptography: RSA was introduced in 1977, while elliptic-curve cryptography was introduced in 1985. However, RSA has shown many more weaknesses than elliptic-curve cryptography. RSA's effective security level was dramatically reduced by the linear sieve in the late 1970s, by the quadratic sieve and ECM in the 1980s, and by the number-field sieve in the 1990s. For comparison, a few attacks have been developed against some rare elliptic curves having special algebraic structures, and the amount of computer power available to attackers has predictably increased, but typical elliptic curves require just as much computer power to break today as they required twenty years ago.

IEEE P1363 standardized elliptic-curve cryptography in the late 1990s, including a stringent list of security criteria for elliptic curves. NIST used the IEEE P1363 criteria to select fifteen specific elliptic curves at five different security levels. In 2005, NSA issued a new "Suite B" standard, recommending the NIST elliptic curves (at two specific security levels) for all public-key cryptography and withdrawing previous recommendations of RSA.

Curve25519, the particular elliptic curve used in NaCl, was introduced in [7] in 2006. It follows all of the standard IEEE P1363 security criteria; it *also* satisfies new recommendations for "twist security" and "Montgomery representation" and "Edwards representation". What this means is that secure implementations of Curve25519 are considerably simpler and faster than secure implementations of (e.g.) NIST P-256; there are fewer opportunities for implementors to make mistakes that compromise security, and mistakes are more easily caught by reviewers.

Montgomery representation allows fast single-scalar multiplication using a Montgomery ladder [29]; this is the bottleneck in Diffie–Hellman key exchange inside `crypto_box`. It was proven in [7] that this scalar-multiplication strategy removes all need to check for special cases inside elliptic-curve additions. NaCl uses a ladder of fixed length to eliminate higher-level branches. Edwards representation allows fast multi-scalar multiplication and general addition with the same advantage of not having to check for special cases. The fixed-base-point scalar multiplication involved in `crypto_sign` uses Edwards representation for additions, and eliminates higher-level branches by using a fixed sequence of 63 point additions as described in [11, Section 4].

Salsa20 [8] is a 20-round 256-bit cipher that was submitted to eSTREAM, the ECRYPT Stream Cipher Project [21], in 2005. The same project collected dozens of submissions from 97 cryptographers in 19 countries, and then hundreds of papers analyzing the submissions. Four refereed papers from 14 cryptographers studied Salsa20, culminating in a 2^{151}-operation "attack" against 7 rounds and a 2^{249}-operation "attack" against 8 rounds. After 3 years of review the eSTREAM committee selected a portfolio of 4 software ciphers, including Salsa20; they recommended 12 rounds of Salsa20 as having a "comfortable margin for security".

For comparison, AES is a 14-round 256-bit cipher that was standardized ten years ago. Cryptanalysis at the time culminated in a 2^{140}-operation "attack" against 7 rounds and a 2^{204}-operation "attack" against 8 rounds. New research in 2011 reported a 2^{254}-operation "attack" against all 14 rounds, marginally exploiting the slow key expansion of AES, an issue that was avoided in newer designs such as Salsa20. (Salsa20 also has no penalty for switching keys.) Overall each round of Salsa20 appears to have similar security to each round of AES, and 20 rounds of Salsa20 provide a very solid security margin, despite being faster than 14 rounds of AES on most CPUs.

A further difficulty with AES is that it relies on lookup tables for high-speed implementations; avoiding lookup tables compromises the speed of AES on most CPUs. Recall that, as discussed in Section 3, NaCl prohibits loading data from secret addresses. We do not mean to say that AES cannot be implemented securely: the NaCl implementation of AES is the bitsliced assembly-language implementation described in [26], together with a portable C implementation following the same approach. However, we are concerned about the extent to which security for AES requires compromising speed. Salsa20 avoids these issues: it avoids all use of lookup tables.

Poly1305 is an information-theoretically secure message-authentication code introduced in [6]. Using Poly1305 with Salsa20 is guaranteed to be as secure as using Salsa20 alone, with a security gap of at most 2^{-106} per byte: an attacker who can break the Poly1305 authentication can also break Salsa20. HMAC does not offer a comparable guarantee.

EdDSA was introduced quite recently in [11]. It is much newer than other primitives in NaCl but is within a well-known cloud of signature systems that includes ElGamal, Schnorr, ECDSA, etc.; it combines the safest choices available

within that cloud. EdDSA is like Schnorr and unlike ECDSA in that it diversifies the hash input, adding resilience against hash collisions, and in that it avoids inversions, simplifying and accelerating implementations. EdDSA differs from Schnorr in using a double-size hash function, further reducing the risk of any hash-function problems; in requiring Edwards curves, again simplifying and accelerating implementations; and in including the public key as a further input to the hash function, alleviating concerns regarding attacks targeting many keys at once. EdDSA also avoids a minor compression mechanism, as discussed in [11]; the compression mechanism is public, so it cannot improve security, and skipping it is essential for EdDSA's fast batch verification. Finally, EdDSA generates per-message secret nonces by hashing each message together with a long-term secret, rather than requiring new randomness for each message.

NaCl's implementation of `crypto_sign` *does* use lookup tables but nevertheless avoids secret indices: each lookup from the table loads all table entries and uses arithmetic to obtain the right value. For details see [11, Section 4]. NaCl's signature verification uses signed-sliding-window scalar multiplication, which takes different amounts of time depending on the scalars, but this does not create security problems and does not violate NaCl's prohibition on secret branches: the scalars are not secret.

To summarize, all of these cryptographic choices are quite conservative. We do not expect any of them to be broken until someone succeeds in building a large quantum computer; before that happens we will extend NaCl to support post-quantum cryptography.

References

1. Albrecht, M.R., Paterson, K.G., Watson, G.J.: Plaintext recovery attacks against SSH. In: Evans, D., Myers, A. (eds.) 2009 IEEE Symposium on Security and Privacy, Proceedings, pp. 16–26. IEEE Computer Society (2009), http://www.isg.rhul.ac.uk/~kp/SandPfinal.pdf 3
2. Alfardan, N.J., Paterson, K.G.: Plaintext-recovery attacks against datagram TLS. In: NDSS 2012 (to appear, 2012), http://www.isg.rhul.ac.uk/~kp/dtls.pdf 3
3. Bacelar Almeida, J., Barbosa, M., Pinto, J.S., Vieira, B.: Formal verification of side channel countermeasures using self-composition. Science of Computer Programming (to appear), http://dx.doi.org/10.1016/j.scico.2011.10.008 3
4. Apple. iPhone end user licence agreement. Copy distributed inside each iPhone 4; transcribed at http://rxt3ch.wordpress.com/2011/09/27/iphone-end-user-liscence-agreement-quick-refrence/ 1
5. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management—part 1: General (revised). NIST Special Publication 800-57 (2007), http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf 4
6. Bernstein, D.J.: The Poly1305-AES message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) Fast Software Encryption. LNCS, vol. 3557, pp. 32–49. Springer (2005), http://cr.yp.to/papers.html#poly1305 5

7. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography—PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer (2006), http://cr.yp.to/papers.html#curve25519 4, 5
8. Bernstein, D.J.: The Salsa20 family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New stream cipher designs: the eSTREAM finalists. LNCS, vol. 4986, pp. 84–97. Springer (2008), http://cr.yp.to/papers.html#salsafamily 5
9. Bernstein, D.J.: DNSCurve: Usable security for DNS (2009), http://dnscurve.org/ 2
10. Bernstein, D.J.: CurveCP: Usable security for the Internet (2011), http://curvecp.org/ 2
11. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer (2011), http://eprint.iacr.org/2011/368 1, 4, 5
12. Bernstein, D.J., Lange, T. (eds.): eBACS: ECRYPT benchmarking of cryptographic systems, http://bench.cr.yp.to 3, 4
13. Bernstein, D.J., Schwabe, P.: NEON crypto. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems: CHES 2012. LNCS, vol. 7428, pp. 320–339. Springer (2012), http://cr.yp.to/papers.html#neoncrypto 4
14. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1. In: Krawczyk, H. (ed.) Advances in Cryptology—CRYPTO '98. LNCS, vol. 1462, pp. 1–12. Springer (1998), http://www.bell-labs.com/user/bleichen/papers/pkcs.ps 3
15. Brumley, B.B., Tuveri, N.: Remote timing attacks are still practical. In: Atluri, V., Diaz, C. (eds.) Computer Security—ESORICS 2011. LNCS, vol. 6879, pp. 355–371. Springer (2011), http://eprint.iacr.org/2011/232/ 3
16. "Bushing", Hector Martin "marcan" Cantero, Boessenkool, S., Peter, S.: PS3 epic fail (2010), http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf 3
17. Chandramouli, R., Rose, S.: Secure domain name system (DNS) deployment guide. NIST Special Publication 800-81r1 (2010), http://csrc.nist.gov/publications/nistpubs/800-81r1/sp-800-81r1.pdf 4
18. Daemen, J., Rijmen, V.: AES proposal: Rijndael, version 2 (1999), http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf 3
19. Dempsky, M.: OpenDNS adopts DNSCurve, http://blog.opendns.com/2010/02/23/opendns-dnscurve/ 2
20. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A strengthened version of RIPEMD. In: Gollmann, D. (ed.) Fast Software Encryption. LNCS, vol. 1039, pp. 71–82. Springer (1996) 5
21. ECRYPT. The eSTREAM project, http://www.ecrypt.eu.org/stream/ 5
22. Gutmann, P.: cryptlib security toolkit, http://www.cs.auckland.ac.nz/~pgut001/cryptlib/ 2
23. Gutmann, P.: cryptlib security toolkit: version 3.4.1: user's guide and manual, ftp://ftp.franken.de/pub/crypt/cryptlib/manual.pdf 2, 3
24. Josefsson, S.: Don't return different errors depending on content of decrypted PKCS#1. Commit to the GnuTLS library (2006), http://git.savannah.gnu.org/gitweb/?p=gnutls.git;a=commit;h=fc43c0d05ac450513b6dcb91949ab03eba49626a 3
25. Kaliski, B.: TWIRL and RSA key size, http://web.archive.org/web/20030618141458/http://rsasecurity.com/rsalabs/technotes/twirl.html 4

26. Käsper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: Clavier, C., Gaj, K. (eds.) Cryptographic Hardware and Embedded Systems—CHES 2009. LNCS, vol. 5747, pp. 1–17. Springer (2009), http://cryptojedi.org/papers/#aesbs 5
27. Langley, A.: ctgrind—checking that functions are constant time with Valgrind (2010), https://github.com/agl/ctgrind 3
28. Molnar, D., Piotrowski, M., Schultz, D., Wagner, D.: The program counter security model: Automatic detection and removal of control-flow side channel attacks. In: Won, D., Kim, S. (eds.) Information Security and Cryptology: ICISC 2005. LNCS, vol. 3935, pp. 156–168. Springer (2005) 3
29. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation 48(177), 243–264 (1987), http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866113-7/S0025-5718-1987-0866113-7.pdf 5
30. OpenSSL. OpenSSL: The open source toolkit for SSL/TLS, http://www.openssl.org/ 1
31. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. In: Pointcheval, D. (ed.) Topics in Cryptology—CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer (2006) 3
32. Shamir, A., Tromer, E.: Factoring large numbers with the TWIRL device. In: Boneh, D. (ed.) Advances in Cryptology—CRYPTO 2003. LNCS, vol. 2729, pp. 1–26. Springer (2003), http://tau.ac.il/~tromer/papers/twirl.pdf 4
33. Smits, I.: QuickTun, http://wiki.ucis.nl/QuickTun 2
34. Software in the Public Interest, Inc. Debian security advisory, DSA-1571-1 openssl—predictable random number generator (2008), http://www.debian.org/security/2008/dsa-1571 3
35. Solworth, J.A.: Ethos: an operating system which creates a culture of security, http://rites.uic.edu/~solworth/ethos.html 2
36. Sotirov, A., Stevens, M., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: MD5 considered harmful today (2008), http://www.win.tue.nl/hashclash/rogue-ca/ 5
37. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collision for MD5 and the creation of a rogue CA certificate. In: Halevi, S. (ed.) Advances in Cryptology—CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer (2009), http://eprint.iacr.org/2009/111/ 5
38. Tor project: Anonymity online, https://www.torproject.org/ 2, 4
39. Tromer, E., Osvik, D.A., Shamir, A.: Efficient cache attacks on AES, and countermeasures. Journal of Cryptology 23(1), 37–71 (2010) 3
40. Ulevitch, D.: Want to do something that matters? Then read on, http://blog.opendns.com/2012/02/06/dnscrypt-hackers-wanted/ 2

Faster Implementation of Scalar Multiplication on Koblitz Curves

Diego F. Aranha[1], Armando Faz-Hernández[2], Julio López[3], and Francisco Rodríguez-Henríquez[2]

[1] Departament of Computer Science, University of Brasília
dfaranha@unb.br
[2] Computer Science Department, CINVESTAV-IPN
armfaz@computacion.cs.cinvestav.mx, francisco@cs.cinvestav.mx
[3] Institute of Computing, University of Campinas
jlopez@ic.unicamp.br

Abstract. We design a state-of-the-art software implementation of field and elliptic curve arithmetic in standard Koblitz curves at the 128-bit security level. Field arithmetic is carefully crafted by using the best formulae and implementation strategies available, and the increasingly common native support to binary field arithmetic in modern desktop computing platforms. The i-th power of the Frobenius automorphism on Koblitz curves is exploited to obtain new and faster interleaved versions of the well-known τNAF scalar multiplication algorithm. The usage of the $\tau^{\lfloor m/3 \rfloor}$ and $\tau^{\lfloor m/4 \rfloor}$ maps are employed to create analogues of the 3-and 4-dimensional GLV decompositions and in general, the $\lfloor m/s \rfloor$-th power of the Frobenius automorphism is applied as an analogue of an s-dimensional GLV decomposition. The effectiveness of these techniques is illustrated by timing the scalar multiplication operation for fixed, random and multiple points. In particular, our library is able to compute a random point scalar multiplication in just below 10^5 clock cycles, which sets a new speed record across all curves with or without endomorphisms defined over binary or prime fields. The results of our optimized implementation suggest a trade-off between speed, compliance with the published standards and side-channel protection. Finally, we estimate the performance of curve-based cryptographic protocols instantiated using the proposed techniques and compare our results to related work.

Keywords: Efficient software implementation, Koblitz elliptic curves, scalar multiplication.

1 Introduction

Since its introduction in 1985, Elliptic Curve Cryptography (ECC) has become one of the most important and efficient public key cryptosystems in use. Its security is based on the computational intractability of solving discrete logarithm problems over the group formed by the rational points on an elliptic curve.

Anomalous binary curves, also known as Koblitz elliptic curves, were introduced in [1]. Since then, these curves have been subject of extensive analysis and

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 177–193, 2012.

study. Given a finite field $\mathbb{F}_q$ for $q = 2^m$, a Koblitz curve $E_a(\mathbb{F}_q)$, is defined as the set of points $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ that satisfy the equation

$$E_a : y^2 + xy = x^3 + ax^2 + 1, \quad a \in \{0, 1\}, \tag{1}$$

together with a point at infinity denoted by $\mathcal{O}$. It is known that $E_a(\mathbb{F}_q)$ forms an additive Abelian group with respect to the elliptic point addition operation. In this paper, E_a is a Koblitz curve with order $\#E_a(\mathbb{F}_{2^m}) = 2^{2-a}r$, where r is an odd prime. Let $\langle P \rangle$ be an additively written subgroup in E_a of prime order r, and let k be a positive integer such that $k \in [0, r-1]$. Then, the elliptic curve scalar multiplication operation computes the multiple $Q = kP$, which corresponds to the point resulting of adding P to itself, $k-1$ times. Given r, P and $Q \in \langle P \rangle$, the Elliptic Curve Discrete Logarithm Problem (ECDLP) consists of finding the unique integer k such that $Q = kP$ holds.

Since Koblitz curves are defined over the binary field $\mathbb{F}_2$, the Frobenius map and its inverse naturally extend to an automorphism of the curve denoted by τ. The τ map takes (x, y) to (x^2, y^2) and $\mathcal{O}$ to $\mathcal{O}$. It can been shown that $(x^4, y^4) + 2(x, y) = \mu(x^2, y^2)$ for every (x, y) on E_a, where $\mu = (-1)^{1-a}$. In other words, τ satisfies $\tau^2 + 2 = \mu\tau$. By solving the quadratic equation, we can associate τ with the complex number $\tau = \frac{-1+\sqrt{-7}}{2}$.

Elliptic curve scalar multiplication is the most expensive operation in cryptographic protocols whose security guarantees are based on the ECDLP. Improving the computational efficiency of this operation is a widely studied problem. Across the years, a number of algorithms and techniques providing efficient implementations with higher performance have been proposed [2]. Many research works have focused their efforts on the unknown point scenario, where the base point P is not known in advance and when only one single scalar multiplication is required, as in the case of the Diffie-Hellman key exchange protocol [3,4,5]. However, there are situations where a single scalar multiplication must be performed on fixed base points such as in the case of the key and signature generation procedures of the Elliptic Curve Digital Signature Algorithm (ECDSA) standard. In other scenarios, such as in the ECDSA signature verification, the simultaneous computation of two scalar multiplications (one with unknown point and the other with fixed point) of the form $R = kG + lQ$, is required. Comparatively less research works have studied the latter cases [6,7,8].

In [9,3], authors evaluated the achievable performance of binary elliptic curve arithmetic in the latest 64-bit micro-architectures, presenting a comprehensive analysis of unknown-point scalar multiplication computations on random and Koblitz NIST elliptic curves at the 112-bit and 192-bit security levels. However, for the 128-bit security level they only considered a random curve with side-channel resistant scalar multiplication.[1] This was mainly due to the unavailability of benchmarking data for curves equipped with endomorphisms and the

[1] Scalar multiplication on curve CURVE2251 was implemented in [3] using the Montgomery laddering approach that is naturally protected against first-order side-channel attacks.

performance penalty of halving-based approaches when applied to standardized curves.

In this work we revisit the software serial computation of scalar multiplication on Koblitz curves defined over binary fields. This study includes the computation of the scalar multiplication using unknown and fixed points; and single and simultaneous scalar multiplication computations as required in the generation and verification of discrete-log based digital signatures. We extend the analysis given in [3,9] and further investigate an alternate curve choice to provide a complete picture of the performance scenario, while also showing through operation counting and experimental results that Koblitz curves are still the fastest choice for deploying curve-based cryptography if sufficient native support for binary field arithmetic is available in the target platform and if resistance to software side-channel attacks can be disregarded.

To this end, we adopted several techniques previously proposed by different authors: (i) formulation of binary field arithmetic using vector instructions [10]; (ii) time-memory trade-offs for the evaluation of fixed 2^k-powers in binary fields [11]; (iii) new formulas for polynomial multiplication over $\mathbb{F}_2$ and its extensions [12]; (iv) efficient support for the recently introduced carry-less multiplier [3].

Besides building on these advancements on finite field arithmetic, this paper presents several novel techniques including: (i) improved implementation of width-w τNAF integer recoding; (ii) a new precomputation scheme for small multiples of a random point in a Koblitz curve; (iii) lazy-reduction formulae for mixed addition in binary elliptic curves; (iv) novel interleaving strategies of the τNAF algorithm for scalar multiplication in Koblitz curves via powers of the Frobenius automorphism. We remark that the interleaved techniques proposed in this work can be seen as the effective application for the first time in Koblitz curves of an s-dimensional GLV decomposition. Moreover, in this work only the "tried and tested" Koblitz curve NIST-K283 is considered, providing immediate compatibility and interoperability with standards and existing implementations. Note, however, that several of our techniques are not restricted in any sense to this curve choice, and can therefore be used to accelerate scalar multiplication in other Koblitz curves at different security levels.

Our main implementation result is a speed record for the unknown-point single-core scalar multiplication computation over the NIST-K283 curve in a little less than 10^5 clock cycles. Running on an Intel Core i7-2600K processor clocked at 3.4 GHz we were able to compute a random point scalar multiplication in just 29.18μs. To our knowledge, this is the fastest software implementation of this operation across all curves ever proposed for cryptographic applications at the 128-bit security level.

This document is structured as follows: Section 2 discusses the low-level techniques used for the implementation of field arithmetic and integer recoding. Section 3 presents high-level techniques for arithmetic in the elliptic curve, comprising improved formulas for mixed addition by means of lazy reduction and strategies for speeding up the scalar multiplication computation by using powers

of the Frobenius automorphism. Section 4 illustrates the efficiency of the proposed techniques reporting operation counts and timings for scalar multiplication in the fixed, unknown and multiple point scenarios; and extensively compares the results with related work. Additionally in this section we estimate the performance of signature and key agreement protocols when they are instantiated with Koblitz curves. The final section concludes the paper with perspectives for further performance improvement based on upcoming instruction sets.

2 Low-Level Techniques

Let $f(z)$ be a monic irreducible polynomial of degree m over $\mathbb{F}_2$. Then, the binary extension field $\mathbb{F}_{2^m}$ is isomorphic to $\mathbb{F}_{2^m} \cong \mathbb{F}_2[z]/(f(z))$, i.e., $\mathbb{F}_{2^m}$ is a finite field of characteristic 2, whose elements are the finite set of all the binary polynomials of degree less than m. In order to achieve a security level equivalent to 128-bit AES when working with binary elliptic curves, NIST recommends to choose the field extension $\mathbb{F}_{2^{283}}$, along with the irreducible pentanomial $f(z) = z^{283} + z^{12} + z^7 + z^5 + 1$. In a modern 64-bit computing platform, an element from the field $\mathbb{F}_{2^m}$ represented in canonical basis requires $n_{64} = \lceil \frac{m}{64} \rceil$ processor words, or $n_{64} = 5$ when $m = 283$. In the rest of this section, descriptions of algorithms and formulas will refer to either generic or fixed versions of the binary field, depending on whether or not the optimization is restricted to the choice of $m = 283$.

As mentioned before, in this work we made an extensive use of vector instruction sets present in contemporary desktop processors. The platform model given in Table 1 extends the notation reported in [10]. There is limited support for flexible bitwise shifting in vector registers, because propagation of bits between the two contiguous 64-bit words requires additional operations. Notice that vectorized multiple-precision or intra-digit shifts can always be made faster when the shift amount is a multiple of 8 by means of the memory alignment instruction or the bytewise shift instruction, respectively, and that a simultaneous table lookup mapping 4-bit indexes to bytes can be implemented through the byte shuffling instruction called `PSHUFB` in the SSE instruction set.

Table 1. Relevant vector instructions for the implementation of binary field arithmetic

Mnemonic	Description	SSE
$\otimes$	Carry-less multiplication	`PCLMULQDQ`
$\ll_{\{8}, \gg_{\{8}$	64-bit bitwise shifts	`PSLLQ,PSRLQ`
$\ll_8, \gg_8$	128-bit bytewise shift	`PSLLDQ,PSRLDQ`
$\oplus, \wedge, \vee$	Bitwise `XOR,AND,OR`	`PXOR,PAND,POR`
$\lhd, \rhd$	Memory alignment/Multi-precision shifts	`PALIGNR`

In the following, we provide brief implementation notes on how relevant field arithmetic operations such as, addition, multiplication, squaring, multi-squaring, modular reduction and inversion; and integer width-w τNAF recoding, were implemented.

Addition. It is the simplest operation in a binary field and can employ the exclusive-or instruction with the largest operand size in the target platform. This is particularly beneficial for vector instructions, but according to our experiments, the 128-bit SSE [13] integer instruction proved to be faster than the 256-bit AVX [14] floating-point instruction due to a higher reciprocal throughput [15] when operands are stored into registers.

Multiplication. Field multiplication is the performance-critical arithmetic operation for elliptic curve arithmetic. Given two field elements $a(z), b(z) \in \mathbb{F}_{2^{283}}$ we want to compute a third field element $c(z) = a(z) \cdot b(z) \bmod f(z)$. This can be accomplished by performing two separate steps: first the polynomial multiplication of the two operands $a(z), b(z)$ is evaluated and then the resulting double length polynomial is modular reduced by $f(z)$. From our field element representation, the polynomial multiplication step can be seen as the computation of the product of two $(n_{64} - 1)$-degree polynomials, each with n_{64} 64-bit coefficients. Alternatively, the two operands may also be seen as $(\lceil \frac{n_{64}}{2} \rceil - 1)$-degree polynomials, each with $\lceil \frac{n_{64}}{2} \rceil$ 128-bit coefficients. In the latter case, each term-by-term multiplication can be solved via the standard Karatsuba formula by performing 3 carry-less multiplications. When $n_{64} = 5$, the above approaches require 13 (see [12,16]) and 14 invocations of the carry-less multiplier instruction, respectively. Algorithm 1 below presents our implementation of field multiplication over the field $\mathbb{F}_{2^{283}}$ with 64-bit granularity using the formula given in [12]. The computational complexity of Algorithm 1 is of 13 carry-less multiplications and 32 vector additions, respectively, plus one modular reduction (Alg. 1, step 22) that will be discussed later. The most salient feature of Algorithm 1 is that all the 13 carry-less multiplications have been grouped into one single loop on steps 6-8. This is an attractive feature from a throughput point of view, as it is important to potentially reduce the cost of the carry-less multiplication instruction from 14 to 8 clock cycles in the Intel Sandy Bridge micro-architecture; and from 12 to 7 clock cycles in an AMD Bulldozer [15]. The rationale behind this cost reduction is that the batch execution of independent multiplications directly benefits the micro-architecture pipeline occupancy level. It is worth mentioning that in [3], authors concluded that the 64-bit granular approach tends to consume more resources and complicate register allocation, limiting the natural throughput exhibited by the carry-less multiplication instruction. However, if the digits are stored in an interleaved form (see [17]), these side effects are mitigated and higher throughput can again be achieved.

Squaring and Multi-Squaring. Squaring is a cheap operation in a binary field due to the action of the Frobenius map, consisting of a linear expansion of coefficients. Vectorized approachs using simultaneous table lookups through byte shuffling instructions allow a particularly efficient formulation of the coefficient expansion step [10]. Modular reduction usually is the most expensive step when computing a squaring, especially when $f(z)$ is an *ordinary pentanomial* (see [18]) for the word size. Dealing efficiently with

ordinary pentanomials requires flexible and often not directly supported shifting instructions in the target platform. Multi-squaring is a time-memory trade-off in which a table of $16\lceil \frac{m}{4} \rceil$ field elements allows computing any fixed 2^k power with the cost equivalent of just a few squarings [11]. It is usually the case that the multi-squaring approach becomes faster than repeated squaring, whenever $k \geq 6$ [3]. Contrary to addition, the availability of 256-bit instructions here contributes significantly to a performance increase. This happens because this operation basically consists of a sequence of additions with field elements obtained through a precomputed table stored in main memory.

Algorithm 1. Proposed implementation of multiplication in $\mathbb{F}_{2^{283}}$

Input: $a(z) = a[0..4], b(z) = b[0..4]$.
Output: $c(z) = c[0..4] = a(z) \cdot b(z)$.
Note: Pairs a_i, b_i, c_i, m_i of 64-bit words represent vector registers.

1: **for** $i \leftarrow 0$ **to** 4 **do**
2: $\quad c_i \leftarrow (a[i], b[i])$
3: **end for**
4: $c_5 \leftarrow c_0 \oplus c_1, \quad c_6 \leftarrow c_0 \oplus c_2, \quad c_7 \leftarrow c_2 \oplus c_4, \quad c_8 \leftarrow c_3 \oplus c_4$
5: $c_9 \leftarrow c_3 \oplus c_6, \quad c_{10} \leftarrow c_1 \oplus c_7, \quad c_{11} \leftarrow c_5 \oplus c_8, \quad c_{12} \leftarrow c_2 \oplus c_{11}$
6: **for** $i \leftarrow 0$ **to** 12 **do**
7: $\quad m_i \leftarrow c_i[0] \otimes c_i[1]$
8: **end for**
9: $c_0 \leftarrow m_0, \quad c_8 \leftarrow m_4$
10: $c_1 \leftarrow c_0 \oplus m_1, \quad c_2 \leftarrow c_1 \oplus m_6$
11: $c_1 \leftarrow c_1 \oplus m_5, \quad c_2 \leftarrow c_2 \oplus m_2$
12: $c_7 \leftarrow c_8 \oplus m_3, \; c_6 \leftarrow c_7 \oplus m_7$
13: $c_7 \leftarrow c_7 \oplus m_8, \; c_6 \leftarrow c_6 \oplus m_2$
14: $c_5 \leftarrow m_{11} \oplus m_{12}, c_3 \leftarrow c_5 \oplus m_9$
15: $c_3 \leftarrow c_3 \oplus c_0 \oplus c_{10}$
16: $c_4 \leftarrow c_1 \oplus c_7 \oplus m_9 \oplus m_{10} \oplus m_{12}$
17: $c_5 \leftarrow c_5 \oplus c_2 \oplus c_8 \oplus m_{10}$
18: $c_9 \leftarrow c_7 \ll_8 64$
19: $(c_7, c_5, c_3, c_1) \leftarrow (c_7, c_5, c_3, c_1) \lhd 8$
20: $c_0 \leftarrow c_0 \oplus c_1, \quad c_1 \leftarrow c_2 \oplus c_3, \quad c_2 \leftarrow c_4 \oplus c_5$
21: $c_3 \leftarrow c_6 \oplus c_7, \quad c_4 \leftarrow c_8 \oplus c_9$
22: **return** $c = (c_4, c_3, c_2, c_1, c_0) \bmod f(z)$

Modular Reduction. Efficient modular reduction of a double-length value resulting of a squaring or multiplication operation to a proper field element involves expressing the required shifted additions in terms of the best shifting instructions possible. For the instruction sets available in our target platform, this amounts to converting the highest possible number of shifts to memory alignment instructions or byte-wise shifts. Curve NIST-K283 is defined over an ordinary pentanomial, a particularly inefficient choice for our vector register size. However, by observing that $f(z) = z^{283} + z^{12} + z^7 + z^5 + 1 =$

$z^{283} + (z^7 + 1)(z^5 + 1)$, one can take advantage of this factorization to formulate faster shifted additions. Algorithm 2 presents our explicit scheduling of shift instructions to perform modular reduction in $\mathbb{F}_{2^{283}}$. Suppose that the polynomial c is written as $c = p_1||p_0$ where the polynomial p_0 represent the lower 283 bits of c. The computation of c mod $f(z)$ in Algorithm 2 is performed as follows: in lines 1 to 3, the polynomial p_1 is computed by shifting the vector (c_4, c_3, c_2) to the right exactly 27 bits. Then, in lines 4 to 10, the operation $c + p_1(z^7 + 1)(z^5 + 1)$ is performed, thus getting the vector (c_2, c_1, c_0) . Finally, in lines 11 to 14, the remaining 101 most significant bits of c_2 are reduced, a process that again involves a multiplication by the polynomial $(z^7 + 1)(z^5 + 1)$.

Algorithm 2. Implementation of reduction by $f(z) = z^{283} + (z^7 + 1)(z^5 + 1)$

Input: Double-precision polynomial stored into 128-bit registers $c = (c_4, c_3, c_2, c_1, c_0)$.
Output: Field element c mod $f(z)$ stored into 128-bit registers (c_2, c_1, c_0).

1: $t_2 \leftarrow c_2, t_0 \leftarrow (c_3, c_2) \rhd 64, t_1 \leftarrow (c_4, c_3) \rhd 64$
2: $c_4 \leftarrow c_4 \gg_{\ell 8} 27, c_3 \leftarrow c_3 \gg_{\ell 8} 27, c_3 \leftarrow c_3 \oplus (t_1 \ll_{\ell 8} 37)$
3: $c_2 \leftarrow c_2 \gg_{\ell 8} 27, c_2 \leftarrow c_2 \oplus (t_0 \ll_{\ell 8} 37)$
4: $t_0 \leftarrow (c_4, c_3) \rhd 120, c_4 \leftarrow c_4 \oplus (t_0 \gg_{\ell 8} 1)$
5: $t_1 \leftarrow (c_3, c_2) \rhd 64, c_3 \leftarrow c_3 \oplus (c_3 \ll_{\ell 8} 7) \oplus (t_1 \gg_{\ell 8} 57)$
6: $t_0 \leftarrow c_2 \ll_8 64, c_2 \leftarrow c_2 \oplus (c_2 \ll_{\ell 8} 7) \oplus (t_0 \gg_{\ell 8} 57)$
7: $t_0 \leftarrow (c_4, c_3) \rhd 120, c_4 \leftarrow c_4 \oplus (t_0 \gg_{\ell 8} 3)$
8: $t_1 \leftarrow (c_3, c_2) \rhd 64, c_3 \leftarrow c_3 \oplus (c_3 \ll_{\ell 8} 5) \oplus (t_1 \gg_{\ell 8} 59)$
9: $t_0 \leftarrow c_2 \ll_8 64, c_2 \leftarrow c_2 \oplus (c_2 \ll_{\ell 8} 5) \oplus (t_0 \gg_{\ell 8} 59)$
10: $c_0 \leftarrow c_0 \oplus c_2, c_1 \leftarrow c_1 \oplus c_3, c_2 \leftarrow t_2 \oplus c_4$
11: $t_0 \leftarrow c_4 \gg_{\ell 8} 27$
12: $t_1 \leftarrow t_0 \oplus (t_0 \ll_{\ell 8} 5)$
13: $t_0 \leftarrow t_1 \oplus (t_1 \ll_{\ell 8} 7)$
14: $c_0 \leftarrow c_0 \oplus t_0, c_2 \leftarrow c_2 \wedge$ (0x0000000000000000,0x0000000007FFFFFF)
15: **return** $c = (c_2, c_1, c_0)$

Inversion. The field inversion approach that probably is the friendliest to vector instruction sets is the Itoh-Tsuji inversion [19] that computes the field inverse of a using the identity $a^{-1} = \left(a^{2^{m-1}-1}\right)^2$. The term $a^{2^{m-1}-1}$ is obtained by sequentially computing intermediate terms of the form

$$\left(a^{2^i-1}\right)^{2^j} \cdot a^{2^j-1}. \tag{2}$$

where the exponents $0 \leq i, j \leq m-1$, are elements of the addition chain associated to the exponent $e = m-1$ [20,21]. The shortest addition chain for $e = 282$ has length 12 and is $1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 17 \rightarrow 34 \rightarrow 35 \rightarrow 70 \rightarrow 140 \rightarrow 141 \rightarrow 282$. The computation of the above outlined procedure introduces an important memory cost of storing 4 multi-squaring tables (for computing powers

$2^{17}, 2^{35}, 2^{70}, 2^{141}$), with each table containing $16\lceil \frac{m}{4} \rceil$ field elements. However, several of those tables can be reused in the interleaving approach for scalar multiplication by exploiting powers of the Frobenius automorphism as will be explained in the next section. We note that other approaches for computing multiplicative field inverses, such as a polynomial version of the extended euclidean algorithm, tend to be not so efficient when vectorized mostly because they require intensive shifting of the intermediate values generated by the algorithm.

Integer τNAF Recoding. Solinas [22] presented a τ-adic analogue of the customary *Non-Adjacent Form* (NAF) recoding. An element $\rho \in \mathbb{Z}[\tau]$ is found with $\rho \equiv k \pmod{\frac{\tau^m - 1}{\tau - 1}}$, of as small norm as possible, where for the subgroup of interest, $kP = \rho P$ and a width-w τNAF representation for ρ can be obtained in a way that mimics the usual width-w NAF recoding. As in [22], let us define $\alpha_i = i \bmod \tau^w$ for $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$. A width-$w$ τNAF of a nonzero element k is an expression $k = \sum_{i=0}^{l-1} u_i \tau^i$ where each $u_i \in \{0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}\}$ and $u_{l-1} \neq 0$, and at most one of any consecutive w coefficients is nonzero. Under reasonable assumptions, this procedure outputs an expansion with length $l \leq m+1$. Although the cost of width-w NAF recoding is usually negligible when compared with the overall cost of scalar multiplication, this is not generally the case with Koblitz curves, where integer to width-w τNAF recoding can reach more than 10% of the computational time for computing a scalar multiplication [3]. In this work, the recoding was implemented by employing as much as possible branchless techniques: the branches inside the recoding operation essentially depend on random values, presenting a worst-case scenario for branch prediction and causing severe performance penalties. In addition to that, the code was also completely unrolled to handle only the precision required in the current iteration. Since the magnitude of the involved scalars gets reduced with each iteration, it is suboptimal to perform operations considering the initial full precision. The deterministic nature of the algorithm allows one to know in which precise iteration of the main recoding loop, the most significant word of the intermediate values become zero, which permits to represent these values with one less processor word.

3 High-Level Techniques

In the last section, several notes gave a general description of our algorithmic and implementation choices for field arithmetic. This section describes the higher-level strategies used in the elliptic curve arithmetic layer for increasing the performance of scalar multiplication.

3.1 Exploiting Powers of the Frobenius Automorphism

Scalar multiplication algorithms on Koblitz curves are always tailored to exploit the Frobenius automorphism τ on $E(\mathbb{F}_{2^m})$ given by $\tau(x, y) = (x^2, y^2)$. One such

example is the classic τNAF scalar multiplication algorithm [22] and its width-w window variants. Given $k \in \mathbb{Z}$ and $P \in E(\mathbb{F}_{2^m})$, these methods work by first writing $k = \sum k_i \tau^i$ for $k_i \in \{0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}\}$, with $\alpha_i = i \bmod \tau^w$ for $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$. Then the scalar multiplication is computed as $kP = \sum k_i \tau^i P$.

While powers τ^i of the automophism can be automatically considered endomorphisms in the context of the GLV method [23], this does not bring any performance improvement, since applying these powers to a point has exactly the same cost of iterating the automorphism during a standard execution of the τNAF algorithm. Nevertheless, by employing time-memory trade-offs for computing fixed 2^i-th powers with cost significantly smaller than i consecutive squarings, a map of the form $\tau^{\lfloor m/i \rfloor}$ can now be seen as an endomorphism useful for accelerating scalar multiplication through interleaving strategies. For example, the map $\psi \equiv \tau^{\lfloor m/2 \rfloor}$ allows an interleaved scalar multiplication of two points from the expression $kP = k_1 P + 2^{\lfloor m/2 \rfloor} k_2 P = \sum k_{1,i} \tau^i P + \sum k_{2,i} \tau^i \psi(P)$, saving the computational cost of $\lfloor \frac{m}{2} \rfloor$ applications of the Frobenius, or $3\lfloor \frac{m}{2} \rfloor$ squarings. This might be seen as a modest saving, since squaring in a binary field is often considered a free of cost operation. However, this is not entirely true when working with cumbersome irreducible polynomials that lead to relatively expensive modular reductions. This is exactly the case studied in this work and, to be more precise, it can be said instead that interleaving via the ψ endomorphism saves the computational cost associated to $3\lfloor \frac{m}{2} \rfloor$ modular reductions.

As explained above, the map ψ achieves an analogue of a bidimensional GLV decomposition for a Koblitz curve. Similarly, the usage of the $\tau^{\lfloor m/3 \rfloor}$ and $\tau^{\lfloor m/4 \rfloor}$ maps can be seen as analogues to 3- and 4-dimensional GLV decompositions or, more generally, the $\lfloor m/s \rfloor$-th power of the Frobenius automorphism as an analogue of an s-dimensional GLV decomposition. In our working case where $m = 283$, note that the addition chain for Itoh-Tsuji inversion was already chosen to include $\lfloor m/2 \rfloor$ and $\lfloor m/4 \rfloor$. Thus, exploiting these powers of the automorphism does not imply additional storage costs. Observe that [24,9] already explored this concept to obtain parallel formulations of scalar multiplication in Koblitz curves.

3.2 Lazy-Reduced Mixed Point Addition

The fastest formula for the mixed addition $R = (X_3, Y_3, Z_3)$ of points $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2)$ in binary curves use López-Dahab coordinates [25] and were proposed in [26]. When the a-coefficient of the curve is 0, the formula is given below:

$$\begin{gathered} A = Y_1 + Y_2 \cdot Z_1^2, \quad B = X_1 + X_2 \cdot Z_1, \quad C = B \cdot Z_1 \\ Z_3 = C^2, \quad D = X_2 \cdot Z_3, \quad E = A \cdot C \\ X_3 = E + (A^2 + C \cdot B^2), \quad Y_3 = (D + X_3) \cdot (E + Z_3) + (Y_2 + X_2) \cdot Z_3^2. \end{gathered}$$

Evaluating this formula has a cost of 8 field multiplications, 5 field squarings and 8 additions. It is possible to further save 2 modular reductions when computing

sums of products in the expressions for the coordinates X_3 and Y_3 given above. This technique is called lazy reduction [27] and trades off a modular reduction by a double-length addition. Our working case presents the best conditions for lazy reduction due to the poor choice of the irreducible pentanomial associated to the NIST K-283 elliptic curve, and the high computational efficiency of the field addition operation. It is then possible to evaluate the formula with a cost equivalent to 8 unreduced multiplications, 5 unreduced squarings, 11 modular reductions, and 10 field addditions. This is very similar to the formula proposed in [28], but without introducing any new coordinates to chain unreduced values across sequential additions.

3.3 Scalar Multiplication Algorithm

Algorithm 3 provides a generic interleaved version of the width-w τNAF point multiplication method when the main loop is folded s times by exploring the $\lfloor m/s \rfloor$-th power of the Frobenius automorphism. In comparison with the original algorithm, approximately $3(s-1)\lfloor \frac{m}{s} \rfloor$ field squarings are saved. Notice however, that incrementing the value s also increases the time and space computational cost of constructing the table of base-point multiples performed in Steps 2-5. In the following, the construction of this table of points is referred as *precomputation phase*.

3.4 Precomputation Scheme

The scalar multiplication algorithm presented in Algorithm 3 requires the computation of the set of affine points $P_{0,u} = \alpha_u P$, for $u \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$. Basically, there are two simple approaches to compute this set: use inversion-free addition in projective coordinates and convert all the points at the end to affine coordinates using the Montgomery's simultaneous inversion method; or perform the additions directly in affine coordinates. High inversion-to-multiplication ratios clearly favor the former approach. The latter can be made more viable when the ratio is moderate and simultaneous inversion is employed for computing the denominators in affine addition.

For an illustration of both approaches, assume the choice $w = 5$, and let M, S, A, I be the cost of multiplication, squaring, addition and inversion in $\mathbb{F}_{2^m}$, respectively. Let us consider first the strategy of performing most of the operations in projective coordinates. For the selected value of w, the first four point multiples of the precomputation table given as,

$$\alpha_1 P = P; \alpha_3 P = (\tau^2 - 1)P; \alpha_5 P = (\tau^2 + 1)P; \alpha_7 P = (\tau^3 - 1)P;$$

can be computed in projective coordinates at a cost of three point additions plus three Frobenius operations. However, the last 4 point multiples in the table, namely,

$$\alpha_9 P = (\tau^3 \alpha_5 + 1)P; \ \ \alpha_{11} P = (-\tau^2 \alpha_5 - 1)P;$$
$$\alpha_{13} P = (-\tau^2 \alpha_5 + 1)P; \alpha_{15} P = (-\tau^2 \alpha_5 - \alpha_5)P;$$

Algorithm 3. Interleaved width-w τNAF scalar multiplication using $\tau^{\lfloor m/s \rfloor}$

Input: $k \in \mathbb{Z}, P \in E(\mathbb{F}_{2^m})$, integer s denoting the interleaving factor.
Output: $kP \in E(\mathbb{F}_{2^m})$.

Compute width-w τ-NAF$(k) = \sum_{i=0}^{l-1} u_i \tau^i$
Compute $P_{0,u} = \alpha_u P$, for $u \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
for $i \leftarrow 1$ **to** $(s-1)$ **do**
    Compute $P_{i,u} = \tau^{\lfloor m/s \rfloor} P_{i-1,u}$
end for
$Q \leftarrow \infty$
for $i \leftarrow l-1$ **to** $s\lfloor \frac{m}{s} \rfloor$ **do**
    $Q \leftarrow \tau Q$
    if $u_i \neq 0$ **then**
        Let u be such that $\alpha_u = u_i$ or $\alpha_{-u} = -u_i$
        if $u_i > 0$ **then** $Q \leftarrow Q + P_{0,u}$; **else** $Q \leftarrow Q - P_{0,u}$
    end if
end for
for $i \leftarrow (\lfloor \frac{m}{s} \rfloor - 1)$ **to** 0 **do**
    $Q \leftarrow \tau Q$
    for $j \leftarrow 0$ **to** $(s-1)$ **do**
        if $u_{i+j\lfloor m/s \rfloor} \neq 0$ **then**
            Let u be such that $\alpha_u = u_{i+j\lfloor m/s \rfloor}$ or $\alpha_{-u} = -u_{i+j\lfloor m/s \rfloor}$
            if $u_i > 0$ **then** $Q \leftarrow Q + P_{j,u}$; **else** $Q \leftarrow Q - P_{j,u}$
        end if
    end for
end for
return $Q = (x, y)$

can be only computed until the point $\tau^2 \alpha_5 P$ has been calculated [2]. This situation requires either an expensive conversion to affine coordinates of the point $\tau^2 \alpha_5 P$ or the lower penalty of performing one general instead of a mixed point addition with an associated cost of $(13M + 4S + 9A)$. Hence, it is possible to compute all the required points with just 6 point additions or subtractions, a single general point addition, 6 Frobenius in affine or projective coordinates and a simultaneous conversion of 7 points to affine coordinates. Half of the 6 point additions and subtractions mentioned above are between points in affine coordinates and considering the associated cost of simultaneous Montgomery inversion, each of them has a computational cost of just $(5M + 3S + 8A)$ and one single inversion. Hence, the total precomputation cost for $w = 5$ is given as,

$$\begin{aligned}\text{Proj. Precomputation cost} &= 3 \cdot (5M + 3S + 8A) + 3 \cdot (8M + 5S + 8A) + \\ &\quad 3 \cdot 2S + 3 \cdot 3S + (13M + 4S + 9A) + \\ &\quad 3 \cdot (7-1)M + I + 7 \cdot (2M + S) \\ &= 84M + 50S + 57A + I.\end{aligned}$$

On the other hand, let us consider the second approach where all the additions are directly performed in affine coordinates. Let us recall that one affine addition

costs $2M + S + I + 8A$. Due to the dependency previously mentioned, we have to split all the affine addition computations into two groups $\{\alpha_3 P, \alpha_5 P, \alpha_7 P\}$ and $\{\alpha_9 P, \alpha_{11} P, \alpha_{13} P, \alpha_{15} P\}$, without dependencies. Computing the first group requires 3 affine additions and a simultaneous inversion to obtain 3 line slopes; whereas the second group requires 4 affine additions and a simultaneous inversion to obtain the 4 slopes, for a total of $7 \cdot (2M + S + 8A) + 3(3-1)M + 3(4-1)M + 2I = 29M + 7S + 56A + 2I$. Considering only the dominant multiplications and inversions, the affine precomputation scheme will be faster than the projective precomputation scheme whenever the inversion-to-multiplication ratio is lower than 55, an assumption entirely compatible with the target platform [3].

4 Estimates, Results and Discussion

4.1 Performance Estimates

Now we are in a position to estimate the performance of Algorithm 3 for the values of $m = 283, s = 1, w = 5$. The algorithm executes the precomputation scheme described in the last section, an average of m applications of the Frobenius automorphism, an expected number of $\frac{m}{w+1}$ additions and a final conversion to affine coordinates. This amounts to a cost of about,

$$\begin{aligned} \text{Estimated cost of Algorithm 3} &= 29M + 7S + 56A + 2I + 283 \cdot 3M + \\ &\quad 47 \cdot (8M + 5S + 8A) + (I + 2M + S) \\ &= 407M + 1092S + 3I \end{aligned}$$

For comparison, the current state-of-the-art serial implementation of a random point multiplication, using a 4-dimensional GLV method over a prime curve and the same choice of w, takes 2 inversions, 665 multiplications, 413 squarings and 844 additions in $\mathbb{F}_{p^2}$, where p has approximately 128 bits [29]. By using the latest formula for 5-term polynomial multiplication described in the last section, the scalar multiplication in Koblitz curves is expected to execute $407 \cdot 13 = 5291$ word multiplications, while the GLV-capable prime curve is expected to execute $(665 \cdot 3 + 413 \cdot 2) \cdot 4 = 11284$ word multiplications. This rough comparison means that a scalar multiplication in a Koblitz curve should be considerably faster than a prime curve equipped with endomorphisms if sufficient support to binary field multiplication is present, or even twice faster if support is equivalent to integer multiplication. Although the latency of the fastest carry-less multiplier available (7 cycles at best [15]) is substantially higher than the integer multiplier counterpart (3 cycles [15]), from our analysis above it is still entirely possible that a careful implementation of a Koblitz curve exhibits lower computational cost.

4.2 Experimental Results

In order to illustrate the performance obtained by the proposed techniques, we implemented a library targeted to the Intel Westmere and Sandy Bridge

micro-architectures, focusing our efforts on benefitting from the SSE and AVX instruction sets with the corresponding availability of 128-bit and 256-bit registers. The library was implemented in the C programming language, with vector instructions accessed through their intrinsics interface. Both version 4.7.1 of the GNU C Compiler Suite (GCC) and version 12.1 of the Intel C Compiler (ICC) were used to build the library in a GNU/Linux environment.

Benchmarking was conducted on Intel Core i5-540M and Core i7-2600K processors clocked at 2.5GHz and 3.4 GHz, respectively, following the guidelines provided in the EBACS website [30]. Namely, automatic overclocking, frequency scaling and HyperThreading technologies were disabled to reduce randomness in the results.

Table 2 presents timings and ratios related to the cost of multiplication for the low-level field arithmetic layer of the library, which computes basic operations in the field $\mathbb{F}_{2^{283}}$. Note how modular reduction dominates the cost of squaring and how the moderate inversion-to-multiplication ratios justify the algorithmic choices. Our best timing on Sandy Bridge for unreduced multiplication is 5% faster than the 135 cycles reported in [31], this saving is obtained by a careful implementation of the same polynomial multiplication formula used in [31].

Table 2. Timings given in clock cycles for basic operations in $\mathbb{F}_{2^{283}}$

	Westmere			Sandy Bridge		
Base field operation	GCC	ICC	op/M	GCC	ICC	op/M
Modular reduction	28	28	0.11	20	22	0.15
Unreduced multiplication	159	163	0.89	128	132	0.89
Multiplication	182	182	1.00	142	149	1.00
Squaring	42	39	0.21	28	29	0.18
Multi-Squaring	287	295	1.62	235	243	1.63
Inversion	4,372	4,268	23.45	3,286	3,308	22.20

Table 3 shows the number of clock cycles for elliptic curve operations, such as point addition, Frobenius endomorphism, and point doubling. The latter is shown only to reflect the improvement of using point doubling-free scalar multiplication as is the case in Koblitz curves. Integer recoding is almost 3 times faster than [3,9], even with longer scalars.

Timings reported for scalar multiplication are divided into three scenarios: (i) known point, where the point to be multiplied is already known before the execution of scalar multiplication; (ii) unknown point, the general case, where the input point is not known until scalar multiplication is processed; (iii) double multiplication of a fixed and a random point, a case usually needed for verifying curve-based digital signatures. For the three scenarios, we used interleaved versions of the left-to-right width-w window τNAF scalar multiplication algorithm with different choices of w. We present timings in Table 4. It was verified experimentally that $s = 2$ is the best choice for random and double point multi-

Table 3. Elliptic curve operations on NIST-K283 when points are represented in affine or López-Dahab coordinates [25]

	Westmere			Sandy Bridge		
Elliptic curve operation	GCC	ICC	op/M	GCC	ICC	op/M
Frobenius (Affine)	84	70	0.38	55	55	0.37
Frobenius (LD)	118	115	0.63	85	83	0.55
Doubling (LD)	965	939	5.15	741	764	5.12
Addition (LD Mixed)	1,684	1,650	9.06	1,300	1,336	8.96
Addition (LD General)	2,683	2,643	14.52	2,086	2,145	14.39
Width-w τNAF recoding	4,841	6,652	36.55	3,954	4,693	31.50

plication, providing a speedup of 3-5% over the conventional case $s = 1$, and that $s = 4$ provides a significant performance increase for fixed point multiplication.

Table 4. Scalar multiplication in three different scenarios: fixed, random and multiple points. Timings are given in 10^3 processing cycles.

	Westmere		Sandy Bridge	
Scalar multiplication	GCC	ICC	GCC	ICC
Random point (kP), $w = 5, s = 1$	139.6	135.1	105.3	105.3
Random point (kP), $w = 5, s = 2$	130.9	127.8	99.2	99.7
Fixed point (kG), $w = 8, s = 2$	80.8	79.0	61.5	62.3
Fixed point (kG), $w = 8, s = 4$	72.6	71.7	55.1	55.9
Fixed/random point ($kG + lQ$), $w_G = 6, w_Q = 5, s = 2$	207.8	206.8	157.7	160.8
Fixed/random point ($kG + lQ$), $w_G = 8, w_Q = 5, s = 2$	192.3	190.6	146.3	148.7

4.3 Comparison to Related Work

The previous state-of-the-art implementation at the 128-bit security level on a Sandy Bridge platform achieves a scalar multiplication of a random point on a prime curve in 120,000 clock cycles [29][2]. A side-channel resistant implementation by Bernstein et al. [8] reports 226,872 cycles for computing this operation on Westmere and 194,208 cycles on Sandy Bridge [30]. Another implementation by Hamburg [32] reports 153,000 cycles on Sandy Bridge. Our implementation is up to 49% faster than related works in this scenario, allowing a significant speedup when computing instances of the ECDH key agreement protocol.

Computing curve-based digital signatures usually amounts to scalar multiplication of fixed points. The authors of [8] report a latency of 87,548 cycles to compute this operation on the Westmere and 70,292 cycles on the Sandy Bridge [30] micro-architectures, while using a precomputed table of 256 points. Hamburg [32] implemented this operation on Sandy Bridge in just 52,000 cycles

[2] The authors reported via private communication that the paper will be updated with new results, costing around 99,000 cycles.

with 160 precomputed points. Compared to the first implementation and using the same number of points, our timings are faster by 22%. Comparing to the second implementation while reducing the number of precomputed points to 128, our timings are slower by 15%.

The last scenario to analyze is signature verification, where work [8] reports single signature verification timings of 273,364 cycles on Westmere and 226,516 cycles on Sandy Bridge [30], while reporting significantly improved timings for batch verification. A faster implementation [32] verifies a signature using 32 precomputed points on Sandy Bridge in 165,000 cycles. We obtain speedups between 5% and 35% on this scenario, considering implementations with the same number of points, and leave the possibility of batch verification as a future direction of this work. It is important to stress that our implementation provides a trade-off between side-channel protection and standards compliance. Consequently, it allows faster and interoperable curve-based cryptography when resistance to side channels is not required.

5 Conclusion

In this work, we presented a software implementation of elliptic curve arithmetic in Koblitz curves defined over binary fields. By reusing several low-level techniques recently-introduced by other authors and proposing a number of useful high-level techniques, we obtained state-of-the-art timings for computing scalar multiplication of a random point, modelling a curve-based key agreement protocol. Our implementation also provides a trade-off between execution time and storage overhead for computing digital signatures and significantly improves the time to verify a single signature. We expect our timings to be accelerated further as support to binary field arithmetic improves on modern 64-bit platforms, either through a faster carry-less multiplier or via the 256-bit integer vector instructions from the upcoming AVX2 instruction set. Our computational cost analysis suggests that if the target platform had a binary field multiplication instruction as efficient as integer multiplication, our implementation could still receive a further factor-2 speedup.

References

1. Koblitz, N.: CM-Curves with Good Cryptographic Properties. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 279–287. Springer, Heidelberg (1992)
2. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, Secaucus (2003)
3. Taverne, J., Faz-Hernández, A., Aranha, D.F., Rodríguez-Henríquez, F., Hankerson, D., López, J.: Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. Journal of Cryptographic Engineering 1(3), 187–199 (2011)
4. Longa, P., Gebotys, C.: Efficient Techniques for High-Speed Elliptic Curve Cryptography. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 80–94. Springer, Heidelberg (2010)

5. Gaudry, P., Thomé, E.: The mpFq library and implementing curve-based key exchanges. In: Software Performance Enhancement of Encryption and Decryption (SPEED 2007), pp. 49–64 (2009), http://www.hyperelliptic.org/SPEED/record.pdf
6. Brown, M., Hankerson, D., López, J., Menezes, A.: Software Implementation of the NIST Elliptic Curves Over Prime Fields. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 250–265. Springer, Heidelberg (2001)
7. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for Faster Elliptic Curve Cryptography on a Large Class of Curves. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 518–535. Springer, Heidelberg (2009)
8. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High-Speed High-Security Signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (2011)
9. Taverne, J., Faz-Hernández, A., Aranha, D.F., Rodríguez-Henríquez, F., Hankerson, D., López, J.: Software Implementation of Binary Elliptic Curves: Impact of the Carry-Less Multiplier on Scalar Multiplication. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 108–123. Springer, Heidelberg (2011)
10. Aranha, D.F., López, J., Hankerson, D.: Efficient Software Implementation of Binary Field Arithmetic Using Vector Instruction Sets. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 144–161. Springer, Heidelberg (2010)
11. Bos, J.W., Kleinjung, T., Niederhagen, R., Schwabe, P.: ECC2K-130 on Cell CPUs. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 225–242. Springer, Heidelberg (2010)
12. Cenk, M., Özbudak, F.: Improved Polynomial Multiplication Formulas over $\mathbb{F}_2$ Using Chinese Remainder Theorem. IEEE Trans. Computers 58(4), 572–576 (2009)
13. Intel: Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference (2002), http://www.intel.com
14. Firasta, N., Buxton, M., Jinbo, P., Nasri, K., Kuo, S.: Intel AVX: New frontiers in performance improvement and energy efficiency (2008), White paper available at http://software.intel.com/
15. Fog, A.: Instruction tables: List of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs (2012), http://www.agner.org/optimize/instruction_tables.pdf
16. Montgomery, P.: Five, six, and seven-term Karatsuba-like formulae. IEEE Transactions on Computers 54(3), 362–369 (2005)
17. Gaudry, P., Brent, R., Zimmermann, P., Thomé, E.: The gf2x binary field multiplication library, https://gforge.inria.fr/projects/gf2x/
18. Scott, M.: Optimal Irreducible Polynomials for $GF(2^m)$ Arithmetic. Cryptology ePrint Archive, Report 2007/192 (2007), http://eprint.iacr.org/
19. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. Inf. Comput. 78(3), 171–177 (1988)
20. Guajardo, J., Paar, C.: Itoh-Tsujii inversion in standard basis and its application in cryptography and codes. Designs, Codes and Cryptography 25(2), 207–216 (2002)
21. Rodríguez-Henríquez, F., Morales-Luna, G., Saqib, N.A., Cruz-Cortés, N.: Parallel Itoh—Tsujii multiplicative inversion algorithm for a special class of trinomials. Des. Codes Cryptography 45(1), 19–37 (2007)
22. Solinas, J.A.: Efficient Arithmetic on Koblitz Curves. Designs, Codes and Cryptography 19(2-3), 195–249 (2000)

23. Gallant, R., Lambert, R., Vanstone, S.: Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001)
24. Ahmadi, O., Hankerson, D., Rodríguez-Henríquez, F.: Parallel formulations of scalar multiplication on Koblitz curves. Journal of Universal Computer Science 14(3), 481–504 (2008)
25. López, J., Dahab, R.: Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n). In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 201–212. Springer, Heidelberg (1999)
26. Al-Daoud, E., Mahmod, R., Rushdan, M., Kiliçman, A.: A New Addition Formula for Elliptic Curves over GF(2^n). IEEE Trans. Computers 51(8), 972–975 (2002)
27. Weber, D., Denny, T.: The Solution of McCurley's Discrete Log Challenge. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 458–471. Springer, Heidelberg (1998)
28. Kim, K.H., Kim, S.I.: A new method for speeding up arithmetic on elliptic curves over binary fields. Cryptology ePrint Archive, Report 2007/181 (2007), http://eprint.iacr.org/
29. Birkner, P., Longa, P., Sica, F.: Four-Dimensional Gallant-Lambert-Vanstone Scalar Multiplication. Cryptology ePrint Archive, Report 2011/608 (2011), http://eprint.iacr.org/, http://www.patricklonga.bravehost.com/speed_ecc.html#speed
30. Bernstein, D.J., Lange, T. (eds.): eBACS: ECRYPT Benchmarking of Cryptographic Systems (May 18, 2012), http://bench.cr.yp.to
31. Su, C., Fan, H.: Impact of Intel's new instruction sets on software implementation of GF(2)[x] multiplication. Inf. Process. Lett. 112(12), 497–502 (2012)
32. Hamburg, M.: Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309 (2012), http://eprint.iacr.org/

Zero-Knowledge for Multivariate Polynomials

Valérie Nachef[1], Jacques Patarin[2], and Emmanuel Volte[1]

[1] Department of Mathematics, University of Cergy-Pontoise, CNRS UMR 8088
2 avenue Adolphe Chauvin, 95011 Cergy-Pontoise Cedex, France
[2] PRISM, University of Versailles
45 avenue des Etats-Unis, 78035 Versailles Cedex, France
{valerie.nachef,emmanuel.volte}@u-cergy.fr,
jacques.patarin@prism.uvsq.fr

Abstract. In [15] a protocol $ZK(2)$ using zero-knowledge argument of knowledge was designed from a solution of a set of multivariate quadratic equations over a finite field (i.e. from MQ problem). In this paper, we propose a new scheme $ZK(d)$ which is a generalization of $ZK(2)$, i.e. we consider systems of polynomials of degree d. The key idea of the scheme $ZK(d)$ is to use a polarization identity that allows to get a d-linear function and then use a cut-and-choose technique. We also observe that the scheme $\tilde{ZK}(d)$, which is the natural generalization of the protocol based on the MQ problem to higher degree, is more efficient in terms of computations whereas the $ZK(d)$ scheme is better in terms of bits to be sent. Moreover these properties are still true for all kinds of polynomials: for example if the polynomials are sparse or dense. Finally, we will present two examples of applications: with Brent equations, or with morphisms of polynomials.

Keywords: Authentication scheme, Zero-Knowledge, Multivariate polynomials.

1 Introduction

The first protocols using zero-knowledge arguments of knowledge were based on the factorization problem, for example Fischer-Micali-Rackoff in 1984, or Fiat-Shamir in 1986, or the Graph Isomorphism Problem. However the factorization problem is not expected to be an NP complete problem (since it is in NP and Co NP) and it has sub-exponential algorithms (such as NFS) and even polynomial algorithms on quantum computers (Shor algorithm). Then, it was proved in 1991 by O. Goldreich, S. Micali and A. Widgerson that any problem of NP has a zero-knowledge proof [6]. But the general construction (cf [6]) of zero-knowledge proofs of knowledge from any problem of NP is usually not very efficient. This is why various zero-knowledge proof of knowledge have been specifically designed from some well suited and well chosen NP complete schemes based on simple combinatorial problems expected to be exponentially difficult, such as PKP of Adi Shamir [16], PP of David Pointcheval [13] or CLE [18] or SD [17] of Jacques Stern for example. In [15] such a scheme was designed from the MQ problem,

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 194–213, 2012.

i.e. the problem of finding a solution from a set of multivariate quadratic equations over a finite field. This MQ problem is related to various primitives in cryptography [2,8,10,11], and is NP-complete over any finite field [4,12].

However, in some cases, it is really interesting to manipulate higher degree systems. For example, when the systems are sparse, or when non random equations with high degree arise naturally like with Brent equations as we will see in this paper. Moreover, the evolution of the algorithms that solve multivariate polynomial equations (we will define this problem as the "MPol" problem) for example with Gröbner basis might imply that we need in the future to make different choices regarding the number of variables, the degree, the field or the type of equation.

This problem was also studied independently of us, and recently in [14] the author designs a public-key identification scheme based on multivariate cubic polynomials. But his technique can not be generalized to multivariate polynomials of degree $d \geq 4$. For example in [14] p. 187 it is written: "*An open problem.* Efficient constructions based on multivariate polynomials of degree ≥ 4 remains as an open problem. However it might be difficult to construct them by using techniques similar to those of [15] or of ours".

In this paper, we will generalize the construction of [15] in order to design a zero-knowledge argument of knowledge from a solution of any set of multivariate polynomials of degree d over a finite field (i.e. not only $d = 2$ or $d = 3$). We consider two schemes extending the results of [15] and [14]. First we define the scheme called $ZK(d)$, for which the key new idea is to use a polarization identity that allows to consider any degree d. In [14] the author uses a linearization of one of the variables. Our polarization technique is more general. Our second scheme, called $\tilde{ZK}(d)$, is the natural generalization of the protocol based on the MQ problem to higher degree. This scheme is "quasi-optimal" in terms of computations, i.e. the number of computations is just proportional to the time to compute the polynomials on the given point. However this scheme was mentioned independently of us in [14] and it was pointed out that it is not efficient (in terms of bits to be sent).

For practical applications the case $d = 3$ (i.e. cubic equations) is particularly important, since from these polynomials we will be able to design zero-knowledge argument of knowledge based on the (NP-complete) Morphism problem (MP) or from the Brent equations related to the optimal way to solve sets of linear equations (i.e. improvements of the Gauss elimination). We will explain in this paper why these two problems are really interesting for cryptography. We can notice that MP (morphism of Polynomial) is NP hard while IP (isomorphism of Polynomials) is expected not to be NP hard (since it has an Arthur-Merlin game for yes or no answers).

2 Zero-Knowledge Arguments of Knowledge and Commitments

In an interactive protocol, there are two entities: the prover and the verifier. The prover wants to convince the verifier that she knows a secret. Both interact

and at the end, the verifier accepts or refuses. In zero-knowledge argument of knowledge, there is a possibility of fraud. A cheater will be able to answer some of the questions (but not all of them). The zero-knowledge arguments of knowledge must be designed such that an answer to one of the questions does not give any indication on the secret but if someone is able to answer all the questions then this will reveal the prover's secret. Informally, the security requirements of a zero-knowledge argument of knowledge are the following (for a more formal reference, see [5]):

1. The zero-knowledge argument of knowledge has **perfect correctness** if a legitimate prover is always accepted.
2. The zero-knowledge argument of knowledge is **statistically zero knowledge** if there exists an efficient simulating algorithm U such that for every feasible Verifier strategy V, the distributions produced by the simulator and the proof protocol are statistically indistinguishable.
3. The zero-knowledge argument of knowledge has **knowledge error** α if there is a knowledge extractor K and a polynomial Q such that if p denotes the probability that K finds a valid witness for x using its access to a prover P^* and p_x denotes the probability that P^* convinces the honest verifier on x, and $p_x > \alpha$, then we have $p \geq Q(p_x - \alpha)$. We emphasize that our scheme has only computational soundness, i.e. the existence of the knowledge extractor will be based on some computational assumption.

In our zero-knowledge arguments of knowledge, we will need string commitment schemes. A string commitment function is denoted by Com. The commitment scheme runs in two phases. In the first phase, the sender computes a commitment value $c = Com(s; \rho)$ and sends c to the receiver, where s is the committed string and ρ is a random string. In the second phase, the sender gives (s, ρ) and the receiver verifies if $c = Com(s; \rho)$. We require the two security properties, statistically hiding and computationally binding. Informally, the first property means that, at the end of the first phase, no receiver can distinguish two commitments values generated from two different strings even if the receiver is computationally unbounded. The second property means that no polynomial-time sender can change the committed string after the first phase. The formal definition and a practical construction of such a commitment is given in [7]. We will assume the existence of such a commitment scheme which can be constructed from a collision resistant hash function [7].

3 Systems of Multivariate Equations of Degree d and the Family of Functions $\mathcal{MPol}(d, n, m, \mathbb{F}_q)$

We denote by MPol(d) the problem of finding at least one solution for a system of multivariate polynomials of degree d over finite field. Then we have MPol(2)=MQ for the quadratic case and MPol(3)=MC for the cubic case. More generally, we will denote simply by "MPol", the problem of finding at least one solution for a system of multivariate polynomials over finite fields. In "MPol" the degree d is

not necessary a critical parameter, for example if there is only one monomial of degree d.

We consider the following function of degree d from $\mathbb{F}_q^n$ to $\mathbb{F}_q^m$:

$$\boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$$

where $\forall \ell,\ 1 \leq \ell \leq m$, and $\boldsymbol{x} = (x_1, \ldots, x_n)$:

$$f_\ell(\boldsymbol{x}) = \sum_{1 \leq i_1 \leq \ldots \leq i_d \leq n} \gamma^\ell_{i_1 \ldots i_d} x_{i_1} x_{i_2} \ldots x_{i_d} + \sum_{1 \leq i_1 \leq \ldots \leq i_{d-1} \leq n} \gamma^\ell_{i_1 \ldots i_{d-1}} x_{i_1} x_{i_2} \ldots x_{i_{d-1}}$$

$$+ \ldots + \sum_{1 \leq i_1 \leq i_2 \leq n} \gamma^\ell_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{1 \leq i_1 \leq n} \gamma^\ell_{i_1} x_{i_1}.$$

We omit the constant term since we are going to deal with a system of the form $\boldsymbol{F}(\boldsymbol{v}) = \boldsymbol{s}$.

We denote by $\mathcal{MP}ol(d, n, m, \mathbb{F}_q)$, the family of the functions $\boldsymbol{F}$ defined above. For example, for quadratic polynomials, the family is denoted by $\mathcal{MP}ol(2, n, m, \mathbb{F}_q)$. In [15], the authors used the notation $\mathcal{MQ}(n, m, \mathbb{F}_q)$.

Let the function $\boldsymbol{G}$ defined from $(\mathbb{F}_q^n)^d$ to $\mathbb{F}_q^m$ by:

$$\boldsymbol{G}(\boldsymbol{r_0}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-1}}) = \sum_{i=1}^{d} (-1)^{d-i} \sum_{\substack{S \subset \{0,\ldots,d-1\} \\ |S|=i}} \boldsymbol{F}(\sum_{j \in S} \boldsymbol{r_j}). \qquad (*)$$

This polarization identity $(*)$ which gives the function $\boldsymbol{G}$ is the keystone of the generalization to any degree d.

For example, when $d = 3$ we have: $\boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$ where $\forall \ell,\ 1 \leq \ell \leq m$ and $\boldsymbol{x} = (x_1, \ldots, x_n)$

$$f_\ell(\boldsymbol{x}) = \sum_{1 \leq i \leq j \leq k \leq n} \gamma^\ell_{ijk} x_i x_j x_k + \sum_{1 \leq i \leq j \leq n} \gamma^\ell_{ij} x_i x_j + \sum_{1 \leq i \leq n} \gamma^\ell_i x_i.$$

And $\boldsymbol{G}$ is defined by:

$$\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \boldsymbol{F}(\boldsymbol{x}+\boldsymbol{y}+\boldsymbol{z}) - \boldsymbol{F}(\boldsymbol{x}+\boldsymbol{y}) - \boldsymbol{F}(\boldsymbol{x}+\boldsymbol{z}) - \boldsymbol{F}(\boldsymbol{y}+\boldsymbol{z}) + \boldsymbol{F}(\boldsymbol{x}) + \boldsymbol{F}(\boldsymbol{y}) + \boldsymbol{F}(\boldsymbol{z}).$$

If: $\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = (g_1(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}), g_2(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}), \ldots, g_m(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}))$
where $\forall \ell,\ 1 \leq \ell \leq m$, $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\boldsymbol{z} = (z_1, \ldots, z_n)$ then:

$$g_\ell(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sum_{1 \leq i \leq j \leq k \leq n} \gamma^\ell_{ijk} (x_i y_j z_k + x_i y_k z_j + x_j y_i z_k + x_j y_k z_i + x_k y_i z_j + x_k y_j z_i).$$

It's easy to check that $g_\ell(\boldsymbol{x}+\boldsymbol{x'}, \boldsymbol{y}, \boldsymbol{z}) = g_\ell(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) + g_\ell(\boldsymbol{x'}, \boldsymbol{y}, \boldsymbol{z})$. We obtain that $\boldsymbol{G}$ is trilinear. For any d we can prove that $\boldsymbol{G}$ is d-linear (see also [20]).

An intractability assumption for a random instance of $\mathcal{MP}ol(d, n, m, \mathbb{F}_q)$ is defined as follows:

Definition 1. *For polynomially bounded functions $n = n(\lambda)$, $m = m(\lambda)$, and $q = q(\lambda)$, it is said that $\mathcal{MP}ol(d, n, m, \mathbb{F}_q)$ is intractable if there is no polynomial-time algorithm that take $(\boldsymbol{F}, \boldsymbol{v})$ generated via $R \in_R \mathcal{MP}ol(d, n, m, \mathbb{F}_q)$, $\boldsymbol{s} \in \mathbb{F}_q^n$, and $\boldsymbol{v} \longleftarrow \boldsymbol{F}(\boldsymbol{s})$ and finds a preimage $\boldsymbol{s'} \in \mathbb{F}_q^n$ such that $\boldsymbol{F}(\boldsymbol{s'}) = \boldsymbol{v}$ with non-negligible probability $\epsilon(\lambda)$.*

For $\boldsymbol{F}$, a multivariate function of degree d, we define a binary relation $R_{\boldsymbol{F}} = \{(\boldsymbol{v}, \boldsymbol{x}) \in \mathbb{F}_q^m \times \mathbb{F}_q^n;\ \boldsymbol{v} = \boldsymbol{F}(\boldsymbol{x})\}$. Given an instance $\boldsymbol{F} \in \mathcal{MP}ol(d, n, m, \mathbb{F}_q)$ and a vector $\boldsymbol{v} \in \mathbb{F}_q^m$ the MPol(d) problem is finding $\boldsymbol{s} \in \mathbb{F}_q^n$ such that $\boldsymbol{F}(\boldsymbol{s}) = \boldsymbol{v}$, i.e. $\boldsymbol{s} \in R_{\boldsymbol{F}}(\boldsymbol{v})$.

4 $ZK(3)$ Schemes

Here we consider the MPol(3) problem which is denoted MC in [14].

4.1 Techniques for Our Constructions

Our constructions employ the cut-and-choose approach, where a prover first divides her secret into shares and then proves the correctness of some shares depending on the choice of a verifier without revealing the secret itself. The shared secrets are the following: $\boldsymbol{s} = \boldsymbol{r_0} + \boldsymbol{r_1} + \boldsymbol{r_2}$, $\boldsymbol{r_0} = \boldsymbol{t_0} + \boldsymbol{t_1}$, $\boldsymbol{F}(\boldsymbol{r_0}) = \boldsymbol{e_0} + \boldsymbol{e_1}$, $\boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_1}) = \boldsymbol{f_0} + \boldsymbol{f_1}$ and $\boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_2}) = \boldsymbol{h_0} + \boldsymbol{h_1}$.

Since $\boldsymbol{G}$ is trilinear we have: $\boldsymbol{G}(\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{r_2}) = \boldsymbol{G}(\boldsymbol{t_0}, \boldsymbol{r_1}, \boldsymbol{r_2}) + \boldsymbol{G}(\boldsymbol{t_1}, \boldsymbol{r_1}, \boldsymbol{r_2})$. Moreover:

$$\begin{aligned}\boldsymbol{G}(\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{r_2}) &= \boldsymbol{F}(\boldsymbol{s}) - \boldsymbol{F}(\boldsymbol{r_0}+\boldsymbol{r_1}) - \boldsymbol{F}(\boldsymbol{r_0}+\boldsymbol{r_2}) - \boldsymbol{F}(\boldsymbol{r_1}+\boldsymbol{r_2}) + \boldsymbol{F}(\boldsymbol{r_0}) + \boldsymbol{F}(\boldsymbol{r_1}) + \boldsymbol{F}(\boldsymbol{r_2}) \\ &= \boldsymbol{v} - \boldsymbol{f_0} - \boldsymbol{f_1} - \boldsymbol{h_0} - \boldsymbol{h_1} - \boldsymbol{F}(\boldsymbol{r_1} + \boldsymbol{r_2}) + \boldsymbol{e_0} + \boldsymbol{e_1} + \boldsymbol{F}(\boldsymbol{r_1}) + \boldsymbol{F}(\boldsymbol{r_2}).\end{aligned}$$

So the prover can express $\boldsymbol{v} = \boldsymbol{F}(\boldsymbol{s})$ in terms of $\boldsymbol{G}(\boldsymbol{t_0}, \boldsymbol{r_1}, \boldsymbol{r_2})$, $\boldsymbol{G}(\boldsymbol{t_1}, \boldsymbol{r_1}, \boldsymbol{r_2})$, $\boldsymbol{t_0}$, $\boldsymbol{t_1}$, $\boldsymbol{e_0}$, $\boldsymbol{e_1}$, $\boldsymbol{f_0}$, $\boldsymbol{f_1}$, $\boldsymbol{h_0}$, $\boldsymbol{h_1}$ and reveals it piece by piece so that $\boldsymbol{r_0}$, $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are never revealed together. We must care that if, for example, $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are revealed together, we should reveal no information on $\boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_1})$ nor on $\boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_2})$ since it could reveal something about $\boldsymbol{r_0}$.

The basic idea is that a prover proves that she has a tuple $(\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{e_0}, \boldsymbol{e_1}, \boldsymbol{f_0}, \boldsymbol{f_1}, \boldsymbol{h_0}, \boldsymbol{h_1})$ satisfying the five equalities:

$$\boldsymbol{t_0} = \boldsymbol{r_0} - \boldsymbol{t_1} \tag{1}$$

$$\boldsymbol{e_0} = \boldsymbol{F}(\boldsymbol{r_0}) - \boldsymbol{e_1} \tag{2}$$

$$\boldsymbol{f_0} = \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_1}) - \boldsymbol{f_1} \tag{3}$$

$$\boldsymbol{h_0} = F(\boldsymbol{r_0} + \boldsymbol{r_2}) - \boldsymbol{h_1} \tag{4}$$

$$\begin{aligned}&\boldsymbol{v} - \boldsymbol{G}(\boldsymbol{t_1}, \boldsymbol{r_1}, \boldsymbol{r_2}) + \boldsymbol{e_1} - \boldsymbol{f_1} - \boldsymbol{h_1} \\ &\quad - \boldsymbol{F}(\boldsymbol{r_1} + \boldsymbol{r_2}) + \boldsymbol{F}(\boldsymbol{r_1}) + \boldsymbol{F}(\boldsymbol{r_2}) = \boldsymbol{G}(\boldsymbol{t_0}, \boldsymbol{r_1}, \boldsymbol{r_2}) - \boldsymbol{e_0} + \boldsymbol{f_0} + \boldsymbol{h_0}\end{aligned} \tag{5}$$

4.2 3-Pass Scheme

For simplicity, the random string in Com is not written explicitly. If X is a set, $x \in_R X$ means that x is randomly chosen in X with the uniform distribution.

1. The Prover picks up $\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{t_0} \in_R \mathbb{F}_q^n$ and $\boldsymbol{e_0}, \boldsymbol{f_0}, \boldsymbol{h_0} \in_R \mathbb{F}_q^m$. Then she computes

$$\begin{aligned} &\boldsymbol{r_2} = \boldsymbol{s} - \boldsymbol{r_1} - \boldsymbol{r_0}, \quad \boldsymbol{t_1} = \boldsymbol{r_0} - \boldsymbol{t_0} \\ &\boldsymbol{e_1} = \boldsymbol{F}(\boldsymbol{r_0}) - \boldsymbol{e_0}, \quad \boldsymbol{f_1} = \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_1}) - \boldsymbol{f_0}, \quad \boldsymbol{h_1} = \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_2}) - \boldsymbol{h_0} \end{aligned}$$

   The Prover sends to the Verifier

$$\begin{aligned} &c_0 = Com(\boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{G}(\boldsymbol{t_0}, \boldsymbol{r_1}, \boldsymbol{r_2}) - \boldsymbol{e_0} + \boldsymbol{f_0} + \boldsymbol{h_0}) \\ &c_1 = Com(\boldsymbol{r_1}, \boldsymbol{t_0}, \boldsymbol{e_0}, \boldsymbol{f_0}), \quad c_2 = Com(\boldsymbol{r_1}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{f_1}) \\ &c_3 = Com(\boldsymbol{r_2}, \boldsymbol{t_0}, \boldsymbol{e_0}, \boldsymbol{h_0}), \quad c_4 = Com(\boldsymbol{r_2}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{h_1}) \end{aligned}$$

2. The verifier chooses a query $\mathcal{Q} \in_R \{0, 1, 2, 3\}$ and sends $\mathcal{Q}$ to the prover.
3. The figure 1 summarizes how the verifier deals with the commitments.
   (a) If $\mathcal{Q} = 0$ then the Prover sends $(\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{f_1})$. The Verifier checks if $c_1 = Com(\boldsymbol{r_1}, \boldsymbol{r_0} - \boldsymbol{t_1}, \boldsymbol{F}(\boldsymbol{r_0}) - \boldsymbol{e_1}, \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_1}) - \boldsymbol{f_1})$, $c_2 = Com(\boldsymbol{r_1}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{f_1})$. Here we use (1), (2) and (3).
   (b) If $\mathcal{Q} = 1$ then the Prover sends $(\boldsymbol{r_0}, \boldsymbol{r_2}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{h_1})$. The Verifier checks if $c_3 = Com(\boldsymbol{r_2}, \boldsymbol{r_0} - \boldsymbol{t_1}, \boldsymbol{F}(\boldsymbol{r_0}) - \boldsymbol{e_1}, \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_2}) - \boldsymbol{h_1})$, $c_4 = Com(\boldsymbol{r_2}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{h_1})$. Here we use (1), (2) and (4).
   (c) If $\mathcal{Q} = 2$ then the Prover sends $(\boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{f_1}, \boldsymbol{h_1})$. The Verifier checks if $c_0 = Com(\boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{v} - \boldsymbol{G}(\boldsymbol{t_1}, \boldsymbol{r_1}, \boldsymbol{r_2}) + \boldsymbol{e_1} - \boldsymbol{f_1} - \boldsymbol{h_1} - \boldsymbol{F}(\boldsymbol{r_1} + \boldsymbol{r_2}) + \boldsymbol{F}(\boldsymbol{r_1}) + \boldsymbol{F}(\boldsymbol{r_2}))$, $c_2 = Com(\boldsymbol{r_1}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{f_1})$, $c_4 = Com(\boldsymbol{r_2}, \boldsymbol{t_1}, \boldsymbol{e_1}, \boldsymbol{h_1})$. Here we use (5).
   (d) If $\mathcal{Q} = 3$ then the Prover sends $(\boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{t_0}, \boldsymbol{e_0}, \boldsymbol{f_0}, \boldsymbol{h_0})$. The Verifier checks if $c_0 = Com(\boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{G}(\boldsymbol{t_0}, \boldsymbol{r_1}, \boldsymbol{r_2}) - \boldsymbol{e_0} + \boldsymbol{f_0} + \boldsymbol{h_0})$, $c_1 = Com(\boldsymbol{r_1}, \boldsymbol{t_0}, \boldsymbol{e_0}, \boldsymbol{f_0})$, $c_3 = Com(\boldsymbol{r_2}, \boldsymbol{t_0}, \boldsymbol{e_0}, \boldsymbol{h_0})$.

The verifier outputs 1 if the she gets the correct value in the commitments, 0 otherwise.

4.3 Properties of the 3-Pass Scheme

It is easy to see that the verifier always accepts an interaction with the honest prover. Thus the 3-pass scheme has perfect correctness.

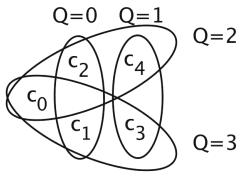

Fig. 1. Check of the commitments

Theorem 1. *The 3-pass protocol is statistically zero knowledge when the commitment scheme Com is statistically hiding.*

Proof. We construct a black-box simulator $\mathcal{S}$ which have oracle access to a cheating verifier $\mathcal{CV}$ takes $\boldsymbol{F}$ and $\boldsymbol{v}$, and outputs a simulated transcripts with probability 3/4 as follows. The simulator randomly chooses a value $\mathcal{Q}^* \in_R \{0,1,2,3\}$ and vectors $\boldsymbol{s'}, \boldsymbol{r'_0}, \boldsymbol{r'_1}, \boldsymbol{t'_0} \in_R \mathbb{F}_q^n$ and $\boldsymbol{e'_0}, \boldsymbol{f'_0}, \boldsymbol{h'_0} \in_R \mathbb{F}_q^m$, where $\mathcal{Q}^*$ is a prediction what value the cheating verifier $\mathcal{CV}$ will not choose. Then it computes

$$\boldsymbol{r'_2} = \boldsymbol{s'} - (\boldsymbol{r'_0} + \boldsymbol{r'_1}),\ \boldsymbol{t'_1} \leftarrow \boldsymbol{r'_0} - \boldsymbol{t'_0},\ \boldsymbol{e'_1} \leftarrow \boldsymbol{F}(\boldsymbol{r'_0}) - \boldsymbol{e'_0}.$$

Moreover it sets in order to simulate the transcripts:

1. If $\mathcal{Q}^* = 0$, $\boldsymbol{f'_1} = \boldsymbol{v} - \boldsymbol{F}(\boldsymbol{s'}) + \boldsymbol{F}(\boldsymbol{r'_0} + \boldsymbol{r'_1}) - \boldsymbol{f'_0}$, else $\boldsymbol{f'_1} = \boldsymbol{F}(\boldsymbol{r'_0} + \boldsymbol{r'_1}) - \boldsymbol{f'_0}$.
2. If $\mathcal{Q}^* = 1$, $\boldsymbol{h'_1} = \boldsymbol{v} - \boldsymbol{F}(\boldsymbol{s'}) + \boldsymbol{F}(\boldsymbol{r'_0} + \boldsymbol{r'_2}) - \boldsymbol{f'_0}$, else $\boldsymbol{h'_1} = \boldsymbol{F}(\boldsymbol{r'_0} + \boldsymbol{r'_2}) - \boldsymbol{h'_0}$.
3. If $\mathcal{Q}^* = 3$, $c'_0 = Com(\boldsymbol{r'_1}, \boldsymbol{r'_2}, \boldsymbol{v} - \boldsymbol{G}(\boldsymbol{t'_1}, \boldsymbol{r'_1}, \boldsymbol{r'_2}) - \boldsymbol{f'_1} - \boldsymbol{h'_1} + \boldsymbol{e'_1} - \boldsymbol{F}(\boldsymbol{r'_1} + \boldsymbol{r'_2}) + \boldsymbol{F}(\boldsymbol{r'_1}) + \boldsymbol{F}(\boldsymbol{r'_2}))$, else $c'_0 = Com(\boldsymbol{r'_1}, \boldsymbol{r'_2}, \boldsymbol{G}(\boldsymbol{t'_0}, \boldsymbol{r'_1}, \boldsymbol{r'_2}) - \boldsymbol{f'_0} - \boldsymbol{h_0} + \boldsymbol{e'_0})$

It also computes:

$$c'_1 = Com(\boldsymbol{r'_1}, \boldsymbol{t'_0}, \boldsymbol{e'_0}, \boldsymbol{f'_0}),\ \ c'_2 = Com(\boldsymbol{r'_1}, \boldsymbol{t'_1}, \boldsymbol{e'_1}, \boldsymbol{f'_1})$$
$$c'_3 = Com(\boldsymbol{r'_2}, \boldsymbol{t'_0}, \boldsymbol{e'_0}, \boldsymbol{h'_0}),\ \ c'_4 = Com(\boldsymbol{r'_2}, \boldsymbol{t'_1}, \boldsymbol{e'_1}, \boldsymbol{h'_1})$$

and sends $(c'_0, c'_1, c'_2, c'_3, c'_4)$ to $\mathcal{CV}$.

Receiving a query $\mathcal{Q}$ from $\mathcal{CV}$ the simulator outputs $\perp$ if $\mathcal{Q} = \mathcal{Q}^*$ and stops. If $\mathcal{S}$ does not output $\perp$, it produces a transcript as follows:

- If $\mathcal{Q} = 0$, it outputs $((c'_0, c'_1, c'_2, c'_3, c'_4), 0, (\boldsymbol{r'_0}, \boldsymbol{r'_1}, \boldsymbol{t'_1}, \boldsymbol{e'_1}, \boldsymbol{f'_1}))$
- If $\mathcal{Q} = 1$, it outputs $((c'_0, c'_1, c'_2, c'_3, c'_4), 1, (\boldsymbol{r'_0}, \boldsymbol{r'_2}, \boldsymbol{t'_1}, \boldsymbol{e'_1}, \boldsymbol{h'_1}))$
- If $\mathcal{Q} = 2$, it outputs $((c'_0, c'_1, c'_2, c'_3, c'_4), 2, (\boldsymbol{r'_1}, \boldsymbol{r'_2}, \boldsymbol{t'_1}, \boldsymbol{e'_1}, \boldsymbol{f'_1}, \boldsymbol{h'_1}))$
- If $\mathcal{Q} = 3$, it outputs $((c'_0, c'_1, c'_2, c'_3, c'_4), 3, (\boldsymbol{r'_1}, \boldsymbol{r'_2}, \boldsymbol{t'_0}, \boldsymbol{e'_0}, \boldsymbol{f'_0}, \boldsymbol{h'_0}))$

We can check that if $\mathcal{S}$ does not output $\perp$, the transcript is accepted. For example, we consider the case where $\mathcal{Q}^* = 0$ and $\mathcal{Q} = 2$. The output is $((c'_0, c'_1, c'_2, c'_3, c'_4), 2, (\boldsymbol{r'_1}, \boldsymbol{r'_2}, \boldsymbol{t'_1}, \boldsymbol{e'_1}, \boldsymbol{f'_1}, \boldsymbol{h'_1}))$. Thus, we have the right values for c'_2 and c'_4. Now, c'_0 is computed as follows: $c'_0 = Com(\boldsymbol{r'_1}, \boldsymbol{r'_2}, \boldsymbol{v} - \boldsymbol{G}(\boldsymbol{t'_1}, \boldsymbol{r'_1}, \boldsymbol{r'_2}) + \boldsymbol{e'_1} - \boldsymbol{f'_1} - \boldsymbol{h'_1} - \boldsymbol{F}(\boldsymbol{r'_1} + \boldsymbol{r'_2}) + \boldsymbol{F}(\boldsymbol{r'_1}) + \boldsymbol{F}(\boldsymbol{r'_2}))$. Here $\boldsymbol{f'_1} = \boldsymbol{v} - \boldsymbol{F}(\boldsymbol{s'}) + \boldsymbol{F}(\boldsymbol{r'_0} + \boldsymbol{r'_1}) - \boldsymbol{f'_0}$. Thus we obtain $c'_0 = Com(\boldsymbol{r'_1}, \boldsymbol{r'_2}, \boldsymbol{G}(\boldsymbol{t'_0}, \boldsymbol{r'_1}, \boldsymbol{r'_2}) - \boldsymbol{f'_0} - \boldsymbol{h_0} + \boldsymbol{e'_0})$ and the transcript is accepted. The other cases are checked similarly.

The distribution of the output $\mathcal{S}$ is statistically close to the distribution of a real transcript since the commitment is statistically hiding and the values are randomly chosen in $\mathbb{F}_q^n$ or in $\mathbb{F}_q^m$. The details can be found in the extended version of the paper. □

Theorem 2. *The 3-pass protocol is proof of zero knowledge with zero knowledge error* 3/4 *when the commitment scheme Com is computationally binding.*

Proof. Suppose that there exists a false prover C that can answer all the questions. Then either C will compute a collision for Com or will extract a solution for $(\boldsymbol{F}, \boldsymbol{v})$. Let $((c_0, c_1, c_2, c_3, c_4), \mathcal{Q}_0, Rsp_0)$, $((c_0, c_1, c_2, c_3, c_4), \mathcal{Q}_1, Rsp_1)$, $((c_0, c_1, c_2, c_3, c_4), \mathcal{Q}_2, Rsp_2)$, $((c_0, c_1, c_2, c_3, c_4), \mathcal{Q}_3, Rsp_3)$, be four transcripts such that $\mathcal{Q}_i = i$ and all the responses are accepted. Consider the situation where the responses are parsed as $Rsp_0 = (\boldsymbol{r_0}^{(0)}, \boldsymbol{r_1}^{(0)}, \boldsymbol{t_1}^{(0)}, \boldsymbol{e_1}^{(0)}, \boldsymbol{f_1}^{(0)})$, $Rsp_1 = (\boldsymbol{r_0}^{(1)}, \boldsymbol{r_2}^{(1)}, \boldsymbol{t_1}^{(1)}, \boldsymbol{e_1}^{(1)}, \boldsymbol{h_1}^{(1)})$, $Rsp_2 = (\boldsymbol{r_1}^{(2)}, \boldsymbol{r_2}^{(2)}, \boldsymbol{t_1}^{(2)}, \boldsymbol{e_1}^{(2)}, \boldsymbol{f_1}^{(2)}, \boldsymbol{h_1}^{(2)})$, $Rsp_3 = (\boldsymbol{r_1}^{(3)}, \boldsymbol{r_2}^{(3)}, \boldsymbol{t_0}, \boldsymbol{e_0}, \boldsymbol{f_0}, \boldsymbol{h_0})$.

We obtain:

$$\begin{aligned} c_0 &= Com(\, \boldsymbol{r_1}^{(2)}, \boldsymbol{r_2}^{(2)}, \boldsymbol{v} - \boldsymbol{G}(\boldsymbol{t_1}^{(2)}, \boldsymbol{r_1}^{(2)}, \boldsymbol{r_2}^{(2)}) - \boldsymbol{f_1}^{(2)} - \boldsymbol{h_1}^{(2)} + \boldsymbol{e_1}^{(2)} \\ &\qquad\qquad -\boldsymbol{F}(\boldsymbol{r_1}^{(2)} + \boldsymbol{r_2}^{(0)}) + \boldsymbol{F}(\boldsymbol{r_1}^{(2)}) + \boldsymbol{F}(\boldsymbol{r_2}^{(2)}) \quad) \\ &= Com(\, \boldsymbol{r_1}^{(3)}, \boldsymbol{r_2}^{(3)}, \boldsymbol{G}(\boldsymbol{t_0}, \boldsymbol{r_1}^{(3)}, \boldsymbol{r_2}^{(3)}) + \boldsymbol{f_0} + \boldsymbol{h_0} - \boldsymbol{e_0} \quad) \qquad (a) \end{aligned}$$

$$\begin{aligned} c_1 &= Com(\, \boldsymbol{r_1}^{(0)}, \boldsymbol{r_0}^{(0)} - \boldsymbol{t_1}^{(0)}, \boldsymbol{F}(\boldsymbol{r_0}^{(0)}) - \boldsymbol{e_1}^{(0)}, \boldsymbol{F}(\boldsymbol{r_0}^{(0)} + \boldsymbol{r_1}^{(0)}) - \boldsymbol{f_1}^{(0)} \,) \\ &= Com(\, \boldsymbol{r_1}^{(3)}, \boldsymbol{t_0}, \quad \boldsymbol{e_0}, \quad \boldsymbol{f_0} \quad) \qquad (b) \end{aligned}$$

$$c_2 = Com(\boldsymbol{r_1}^{(0)}, \boldsymbol{t_1}^{(0)}, \boldsymbol{e_1}^{(0)}, \boldsymbol{f_1}^{(0)}) = Com(\boldsymbol{r_1}^{(2)}, \boldsymbol{t_1}^{(2)}, \boldsymbol{e_1}^{(2)}, \boldsymbol{f_1}^{(2)}) \qquad (c)$$

$$\begin{aligned} c_3 &= Com(\, \boldsymbol{r_2}^{(1)}, \boldsymbol{r_0}^{(1)} - \boldsymbol{t_1}^{(1)}, \boldsymbol{F}(\boldsymbol{r_0}^{(1)}) - \boldsymbol{e_1}^{(1)}, \boldsymbol{F}(\boldsymbol{r_0}^{(1)} + \boldsymbol{r_2}^{(1)}) - \boldsymbol{h_1}^{(1)} \,) \\ &= Com(\, \boldsymbol{r_2}^{(3)}, \boldsymbol{t_0}, \quad \boldsymbol{e_0}, \quad \boldsymbol{h_0} \quad) \qquad (d) \end{aligned}$$

$$c_4 = Com(\boldsymbol{r_2}^{(1)}, \boldsymbol{t_1}^{(1)}, \boldsymbol{e_1}^{(1)}, \boldsymbol{h_1}^{(1)}) = Com(\boldsymbol{r_2}^{(2)}, \boldsymbol{t_1}^{(2)}, \boldsymbol{e_1}^{(2)}, \boldsymbol{h_1}^{(2)}) \qquad (e)$$

If two tuples of the arguments of Com are are distinct on either of the above equations, then we have a collision for Com. Otherwise, these equalities give:

$$\boldsymbol{h_1}^{(1)} \overset{(e)}{=} \boldsymbol{h_1}^{(2)}, \boldsymbol{r_1}^{(0)} \overset{(b)}{=} \boldsymbol{r_1}^{(3)} \overset{(a)}{=} \boldsymbol{r_1}^{(2)}, \boldsymbol{t_1}^{(0)} \overset{(c)}{=} \boldsymbol{t_1}^{(1)} \overset{(e)}{=} \boldsymbol{t_1}^{(2)}, \boldsymbol{r_0}^{(0)} \overset{(c,d)}{=} \boldsymbol{r_0}^{(1)}$$

$$\text{and} \quad \boldsymbol{r_2}^{(2)} \overset{(a)}{=} \boldsymbol{r_2}^{(3)} \overset{(d)}{=} \boldsymbol{r_2}^{(1)}, \boldsymbol{e_1}^{(2)} \overset{(c)}{=} \boldsymbol{e_1}^{(0)} \overset{(b,d)}{=} \boldsymbol{e_1}^{(1)}, \boldsymbol{f_1}^{(0)} \overset{(c)}{=} \boldsymbol{f_1}^{(2)}.$$

So, all upper scripts are useless and from (a) we have:

$$\boldsymbol{v} = \boldsymbol{G}(\boldsymbol{t_1} + \boldsymbol{t_0}, \boldsymbol{r_1}, \boldsymbol{r_2}) + \boldsymbol{f_1} + \boldsymbol{h_1} - \boldsymbol{e_1} + \boldsymbol{F}(\boldsymbol{r_1} + \boldsymbol{r_2}) - \boldsymbol{F}(\boldsymbol{r_1}) - \boldsymbol{F}(\boldsymbol{r_2}) + \boldsymbol{f_0} + \boldsymbol{h_0} - \boldsymbol{e_0}.$$

Then, from (b) and (d) we have $\boldsymbol{r_0} = \boldsymbol{t_0} + \boldsymbol{t_1}$, $\boldsymbol{F}(\boldsymbol{r_0}) = \boldsymbol{e_0} + \boldsymbol{e_1}$, $\boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_1}) = \boldsymbol{f_0} + \boldsymbol{f_1}$ and $\boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_2}) = \boldsymbol{h_0} + \boldsymbol{h_1}$, so if we replace these values in the previous equality, we obtain: $\boldsymbol{v} = \boldsymbol{G}(\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{r_2}) + \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_1}) + \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_2}) + \boldsymbol{F}(\boldsymbol{r_1} + \boldsymbol{r_2}) - \boldsymbol{F}(\boldsymbol{r_0}) - \boldsymbol{F}(\boldsymbol{r_1}) - \boldsymbol{F}(\boldsymbol{r_2}) = \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_1} + \boldsymbol{r_2})$. This means that a solution $\boldsymbol{r_0} + \boldsymbol{r_1} + \boldsymbol{r_2}$ for $\boldsymbol{v}$ is extracted. □

4.4 Computations in the 3-Pass Scheme

We give the maximum number of computations that have to be done either by the prover or by the receiver in the case of $\mathbb{F}_2$. We must calculate the number of computations for F and for G. Moreover we see that F is computed at most

3 times and G is computed just one time. We only count multiplications. In $\mathbb{F}_2$, we have: $x_i^3 = x_i^2 = x_i$. We can write:

$$f_\ell(\boldsymbol{x}) = \sum_{i=1}^{n} x_i \big[\gamma_i^\ell + \gamma_{ii}^\ell + \gamma_{iii}^\ell + \sum_{j=i+1}^{n} x_j \big(\gamma_{ij}^\ell + \gamma_{ijj}^\ell + \sum_{k=j+1}^{n} \gamma_{ijk}^\ell x_k\big)\big]$$

Let M denotes the number of multiplications needed to compute F. Using the above expression for $\boldsymbol{F}$, we obtain $M \simeq \frac{n^3}{6} m$. We want to set the parameters in order to have 80-bit security. Since even for a quadratic system with 84 variables and 80 equations over $\mathbb{F}_2$ satisfies 80-bits security [15], we will use $\mathcal{MP}ol(3, 84, 80, \mathbb{F}_2)$. In the following table, we give the characteristics of the scheme $ZK(3)$ and the values that we obtain when we choose $n = 84$, $m = 80$, in order to have 80-bit security (cf. [2]). Moreover, if we want an impersonation probability less than 2^{-30}, we need to perform at least 73 rounds. R stands for the number of rounds and C for the maximum number of computations that have to be done either by the prover or by the receiver. Moreover, we will use a standard trick for saving communication data size. We employ a collision resistant hash function H. Let $c_\alpha = H(c_1, c_2)$, $c_\beta = H(c_3, c_4)$ and $c = H(c_0, c_\alpha, c_\beta)$. In the first pass, the sender send the value c, instead of five commitments $(c_0, c_1, c_2, c_3, c_4)$. In the third pass, she will proceed as follows:

- If $\mathcal{Q} = 0$, the pair (c_0, c_β) is appended to the prover's responses.
- If $\mathcal{Q} = 1$, the pair (c_0, c_α) is appended to the prover's responses.
- If $\mathcal{Q} = 2$, the pair (c_1, c_3) is appended to the prover's responses.
- If $\mathcal{Q} = 3$, the pair (c_2, c_4) is appended to the prover's responses.

Finally she checks if $c = H(c_0, c_\alpha, c_\beta)$. She can obtain the needed values from the prover's responses in each case. As a result, the number of hash values sent is reduced from 5 to 3. This modified version of the 3-pass protocol has the same properties. Table 1 summarizes the features of $ZK(3)$.

Table 1. $ZK(3)$ Scheme

	Formulas	Parameters for 2^{80} security
Public key (Number of bits)	m	80
Secret key (Number of bits)	n	84
M	$\frac{n^3}{6} \times m$	7902720
Communication (bit)	$(3 \times 160 + 2 + 3n + 3m) \times R$	71102
Number of multiplications	C=9MR	2^{33}

Remarks. We may need less computations. It depends on the number of non zero coefficients. This is the case for Brent equations as explain in Section 8.

It is also possible to design a 5-pass scheme. This in done in the extended version of this paper.

5 $ZK(d)$ Scheme for any d

5.1 The $ZK(d)$ Scheme

We will design a 3-pass scheme.

We consider the following function of degree d from $\mathbb{F}_q^n$ to $\mathbb{F}_q^m$:

$$\boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$$

where $\forall \ell,\ 1 \leq \ell \leq m$,

$$f_\ell(\boldsymbol{x}) = \sum_{1 \leq i_i \leq \ldots \leq i_d \leq n} \gamma^\ell_{i_1 \ldots i_d} x_{i_1} x_{i_2} \ldots x_{i_d} +$$

$$\sum_{1 \leq i_i \leq \ldots \leq i_{d-1} \leq n} \gamma^\ell_{i_1 \ldots i_{d-1}} x_{i_1} x_{i_2} \ldots x_{i_{d-1}} + \ldots + \sum_{1 \leq i_i \leq i_2 \leq n} \gamma^\ell_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{1 \leq i_i \leq n} \gamma^\ell_{i_1} x_{i_1}$$

and $\boldsymbol{x} = (x_1, \ldots, x_n)$. We omit the constant term. Let

$$\boldsymbol{G}(\boldsymbol{r_0}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-1}}) = \sum_{i=1}^{d} (-1)^{d-i} \sum_{\substack{S \subset \{0,\ldots,d-1\} \\ |S|=i}} \boldsymbol{F}(\sum_{j \in S} \boldsymbol{r_j}).$$

The public key is $(\boldsymbol{F}, \boldsymbol{v})$. The secret is $\boldsymbol{s}$ such that $\boldsymbol{F}(\boldsymbol{s}) = \boldsymbol{v}$.

1. The Prover picks up at random $\boldsymbol{r_0}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-2}}, \boldsymbol{t_0} \in_R \mathbb{F}_q^n$, $\boldsymbol{f_0} \in_R \mathbb{F}_q^m$, and $\forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1,\ \boldsymbol{f_0}^{i_1 \ldots i_p} \in_R \mathbb{F}_q^m$. Then she computes

$$\begin{array}{ll} \boldsymbol{r_{d-1}} & = \boldsymbol{s} - \sum_{i=1}^{d-2} \boldsymbol{r_i} \\ \boldsymbol{t_1} & = \boldsymbol{r_0} - \boldsymbol{t_0} \\ \boldsymbol{f_1} & = \boldsymbol{F}(\boldsymbol{r_0}) - \boldsymbol{f_0} \end{array}$$

and

$$\forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1$$

$$\boldsymbol{f_1}^{i_1 \ldots i_p} = \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_{i_1}} + \ldots + \boldsymbol{r_{i_p}}) - \boldsymbol{f_0}^{i_1 \ldots i_p}.$$

Then the Prover sends to the Verifier

$$c_0 \leftarrow Com\Big(\boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{G}(\boldsymbol{t_0}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-1}}) + \sum_{p=1}^{d-2} (-1)^{d-p} \sum_{1 \leq i_1 < \ldots < i_p \leq d-1} \boldsymbol{f_0}^{i_1 \ldots i_p} + (-1)^d \boldsymbol{f_0} \Big)$$

and she sends also $\forall i,\ 1 \leq i \leq d-1$:

$$c_{2i-1} \leftarrow Com\Big(\boldsymbol{r_1}, \ldots, \boldsymbol{r_{i-1}}, \boldsymbol{r_{i+1}}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{t_0}, \boldsymbol{f_0}, \forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1,$$

such that $\forall j,\ i_j \neq i,\ \boldsymbol{f_0}^{i_1 \ldots i_p}\Big)$

$c_{2i} \leftarrow Com\Big(\ \boldsymbol{r_1}, \ldots, \boldsymbol{r_{i-1}}, \boldsymbol{r_{i+1}}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{t_1}, \boldsymbol{f_1},$
$\forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1,$
such that $\forall j\ i_j \neq i,\ \boldsymbol{f_1}^{i_1 \ldots i_p}\Big)$.

The verifier chooses a query $\mathcal{Q} \in_R \{0, 1, \ldots, d\}$ and sends $\mathcal{Q}$ to the prover.

2. (a) If $\mathcal{Q} = 0$, then the Prover sends $(\boldsymbol{r_1}, \boldsymbol{r_2}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{t_0}, \boldsymbol{f_0},$
$\forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1,\ \boldsymbol{f_0}^{i_1 \ldots i_p})$.
The Verifier checks if

$$c_0 = Com\Big(\boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{G}(\boldsymbol{t_0}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-1}}) + \sum_{p=1}^{d-2}(-1)^{d-p} \sum_{1 \leq i_1 < \ldots < i_p \leq d-1} \boldsymbol{f_0}^{i_1 \ldots i_p} + (-1)^d \boldsymbol{f_0}\Big)$$

and $\forall i,\ 1 \leq i \leq d-1$,

$c_{2i-1} = Com\Big(\ \boldsymbol{r_1}, \ldots, \boldsymbol{r_{i-1}}, \boldsymbol{r_{i+1}}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{t_0}, \boldsymbol{f_0},$
$\forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1,$
such that $\forall j,\ i_j \neq i,\ \boldsymbol{f_0}^{i_1 \ldots i_p}\Big)$.

(b) If $\mathcal{Q} = d$, then the Prover sends $(\boldsymbol{r_1}, \boldsymbol{r_2}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{t_1}, \boldsymbol{f_1},$
$\forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1,\ \boldsymbol{f_1}^{i_1 \ldots i_p})$.
$\forall i,\ 1 \leq i \leq d-1$,
The Verifier checks if

$$c_0 = Com\Big(\ \boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{v} - \boldsymbol{G}(\boldsymbol{t_1}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{d-1}}) - \sum_{p=1}^{d-2}(-1)^{d-p} \sum_{1 \leq i_1 < \ldots < i_p \leq d-1} \boldsymbol{f_1}^{i_1 \ldots i_p} - (-1)^d \boldsymbol{f_1} + \sum_{i=1}^{d}(-1)^{d-i} \sum_{\substack{S \subset \{1, \ldots, d-1\} \\ |S| = i}} \boldsymbol{F}(\sum_{j \in S} \boldsymbol{r_j})\Big)$$

and $\forall i,\ 1 \leq i \leq d-1$,

$c_{2i} = Com\Big(\ \boldsymbol{r_1}, \ldots, \boldsymbol{r_{i-1}}, \boldsymbol{r_{i+1}}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{t_1}, \boldsymbol{f_1},$
$\forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1,$
such that $\forall j,\ i_j \neq i,\ \boldsymbol{f_1}^{i_1 \ldots i_p}\Big)$.

(c) if $\mathcal{Q} = i$, then the prover sends $(\boldsymbol{r_0}, \boldsymbol{r_1}, \ldots, \boldsymbol{r_{i-1}}, \boldsymbol{r_{i+1}}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{t_1}, \boldsymbol{f_1}, \forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1,$ such that $\forall j,\ i_j \neq i,\ \boldsymbol{f_1}^{i_1 \ldots i_p})$
The Verifier checks if

$$c_{2i-1} = Com\Big(\boldsymbol{r_1}, \ldots, \boldsymbol{r_{i-1}}, \boldsymbol{r_{i+1}}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{r_0} - \boldsymbol{t_1}, \boldsymbol{F}(\boldsymbol{r_0}) - \boldsymbol{f_1}, \forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1, \text{ such that } \forall j,\ i_j \neq i,\ \boldsymbol{F}(\boldsymbol{r_0} + \boldsymbol{r_{i_1}} + \ldots + \boldsymbol{r_{i_p}}) - \boldsymbol{f_1}^{i_1 \ldots i_p} \Big)$$

and

$$c_{2i} = Com\Big(\boldsymbol{r_1}, \ldots, \boldsymbol{r_{i-1}}, \boldsymbol{r_{i+1}}, \ldots, \boldsymbol{r_{d-1}}, \boldsymbol{t_1}, \boldsymbol{f_1}, \forall p,\ 1 \leq p \leq d-2,\ \forall i_1, \ldots i_p,\ 1 \leq i_1 < i_2 < \ldots i_p \leq d-1, \text{ such that } \forall j,\ i_j \neq i,\ \boldsymbol{f_1}^{i_1 \ldots i_p} \Big).$$

We now give the properties of the $ZK(d)$ Scheme. The proof are detailed in the extended version of this paper.

Theorem 3. *The 3-pass protocol is statistically zero knowledge when the commitment scheme Com is statistically hiding.*

Theorem 4. *The 3-pass protocol is proof of zero knowledge with knowledge error* $\frac{d}{d+1}$ *when the commitment scheme Com is computationally binding.*

We provide the features of this scheme in table 2. The trick used to compute the number of communication bits for $ZK(3)$ can be generalized to $ZK(d)$.

Remark. As for the case $d = 3$, it is possible to design a 5-pass scheme.

6 The $\tilde{ZK}(d)$ Scheme

In this section, we propose another scheme inspired from [15] and also mentioned in [14]. The idea is to transform the system of degree d into a quadratic one and the to use the scheme given in [15]. As we will see, the number of computations is smaller, but the number of communication bits is more important.

First, we investigate the transformation of a system with cubic equations to a system with quadratic equations. We will introduce new variables. Once we have obtained a system of equations with quadratic polynomials, we can apply the identification scheme of [15]. We will calculate the number of multiplications in this case. In our system, we have $\boldsymbol{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$ where $\forall \ell,\ 1 \leq \ell \leq m$,

$$f_\ell(\boldsymbol{x}) = \sum_{1 \leq i \leq j \leq k \leq n} \gamma_{ijk}^\ell x_i x_j x_k + \sum_{1 \leq i \leq j \leq n} \gamma_{ij}^\ell x_i x_j + \sum_{1 \leq i \leq n} \gamma_i^\ell x_i$$

Table 2. $ZK(d)$ scheme

	$ZK(d)$ scheme
Public key (Number of bits)	m
Secret key (Number of bits)	n
M	$\frac{n^d}{d!} \times m$
Rounds	$R = \lceil \frac{30 \ln(2)}{\ln(1+1/d)} \rceil$
Number of Communication (bits)	$(160d + \lceil \frac{\ln d}{2} \rceil + 2^{d-1} - 1) \cdot R$
Multiplications	$C = 9(2^{d-1} - 1 + d!)MR$

and $\boldsymbol{x} = (x_1, \ldots, x_n)$. We introduce the new variables $\forall i, j,\ 1 \leq i \leq j \leq n,\ X_{ij} = x_i x_j$. The number of new variables is $\frac{n(n-1)}{2}$, if $q = 2$, and $\frac{n(n+1)}{2}$, if $q \neq 2$. In our new system, we have

$$\tilde{F} = (\tilde{f}_1, \ldots, \tilde{f}_m, (\tilde{f}_{ij})_{1 \leq i \leq j \leq n})$$

where for $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_n, (X_{ij})_{1 \leq i \leq j \leq n})$ and $1 \leq \ell \leq m$,

$$f_\ell(\tilde{\boldsymbol{x}}) = \sum_{1 \leq i \leq j \leq k \leq n} \gamma^\ell_{ijk} X_{ij} x_k + \sum_{1 \leq i \leq j \leq n} \gamma^\ell_{ij} X_{ij} + \sum_{1 \leq i \leq n} \gamma^\ell_i x_i$$

and for $1 \leq i \leq j \leq n$, $f_{ij}(\tilde{x}) = X_{ij} - x_i x_j$. Here the number of variables is $\tilde{n} \simeq n + \frac{n^2}{2}$ and the number of equations is $\tilde{m} = m + \frac{n^2}{2}$. As before, M denotes the number of multiplications needed to compute F. $\tilde{M}$ denotes the number of multiplications for the computation of $\tilde{F}$. We choose $q = 2$. Then $\tilde{M} = M + \frac{n(n-1)}{2}$. Thus $\tilde{M} \simeq M \simeq \frac{n^3}{6} m$. $\tilde{C}$ stands for the maximum number of computations that have to be done either by the prover or by the receiver. If $\tilde{R}$ denotes the number of rounds performed in order to have an impersonation probability less than 2^{-30}, then $\tilde{R} = 52$ (cf. [15]).

More generally, for $d \geq 4$, we introduce new variables and new equations in order to get a system of quadratic equations. Again there are two functions F and $\tilde{F}$. If M (resp. $\tilde{M}$) denotes the number of computations for F (resp. $\tilde{F}$), then $M \simeq n^d/d!$ and $\tilde{M} \simeq \frac{n^d}{d!} + n^{\lceil \frac{d}{2} \rceil}/\lceil \frac{d}{2} \rceil! \simeq M$. For $ZK(d)$, there are n variables and m equations. For $\tilde{ZK}(d)$, there are $\tilde{n} \simeq n + n^{\lceil \frac{d}{2} \rceil}/\lceil \frac{d}{2} \rceil!$ variables and $\tilde{m} \simeq m + n^{\lceil \frac{d}{2} \rceil}/\lceil \frac{d}{2} \rceil!$ equations. The following table gives the characteristics of the $\tilde{ZK}(d)$ scheme and the values we get when $n = 84$ and $m = 80$.

7 Relations between the Number of Computations and the Number of Coefficients

In the previous computations, we considered the MPol(d) problem since we supposed that we have the maximum number of coefficients. Then we obtained that

Table 3. $\tilde{ZK}(d)$ scheme

	$\tilde{ZK}(3)$	$\tilde{ZK}(d)$	$\tilde{ZK}(3)$
	Formulas		2^{-80} security
Public key (bit)	$\tilde{m}$	$\simeq m + n^{\lceil \frac{d}{2} \rceil}/\lceil \frac{d}{2} \rceil!$	3483
Secret key (bit)	n		84
M	$\frac{n^3}{6} \times m$	$\frac{n^d}{d!} \times m$	7902720
Rounds	$\tilde{R} = 52$		
Communication (bits)	$2 \times 160 + 2 + 2\tilde{n} + \tilde{m}$ $\times \tilde{R}$	$322 + 2n + m + 3n^{\lceil \frac{d}{2} \rceil}/\lceil \frac{d}{2} \rceil!$ $\times \tilde{R}$	560508
Multiplications	$\tilde{C} = 3\tilde{M}\tilde{R}$		2^{31}

in both cases, $M \simeq \tilde{M} \simeq \frac{n^d}{d!}$. Then the total number of multiplications is a function of M or $\tilde{M}$ and the number of rounds. For sparse systems, M or $\tilde{M}$ will be smaller but we will still have the same relations between C, M and R (and similarly $\tilde{C}$, $\tilde{M}$ and $\tilde{R}$). Here again, we can see that there are more variables and more communications bits in the $\tilde{ZK}(d)$ schemes and more computations in the $ZK(d)$ schemes. This is related to the "MPol" problem.

More precisely with the $ZK(d)$ scheme, we have:

$$f_\ell(x_1, x_2, \ldots, x_n) = \sum_{(i_1,\ldots,i_d)\in S_d^\ell} \gamma^\ell_{i_1\ldots i_d} x_{i_1} x_{i_2} \ldots x_{i_d} + \sum_{i_i,\ldots,i_{d-1}\in S_{d-1}^\ell} \gamma^\ell_{i_1\ldots i_{d-1}} x_{i_1} x_{i_2} \ldots x_{i_{d-1}}$$

$$+ \ldots + \sum_{i_i,i_2\in S_2^\ell} \gamma^\ell_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_i\in S_2^\ell} \gamma^\ell_{i_1} x_{i_1}$$

where each S_u^ℓ is a subset of $\{1, 2, \ldots, n\}^u$.

The number of multiplications for f_ℓ is given by

$$d|S_d^\ell| + (d-1)|S_{d-1}^\ell| + (d-2)|S_{d-2}^\ell| + \ldots + 2|S_2^\ell| + |S_1^\ell|$$

and for F the number M of multiplications is

$$M = \sum_{\ell=1}^{m} \Big[d|S_d^\ell| + (d-1)|S_{d-1}^\ell| + (d-2)|S_{d-2}^\ell| + \ldots + 2|S_2^\ell| + |S_1^\ell| \Big].$$

Moreover, F is computed at most $2^{d-1} - 1$ times during the process. For g_l we have $\big(d!(d-1)+1\big)|S_d^\ell|$ multiplications and for G, $\sum_{\ell=1}^{m} \big(d!(d-1)+1\big)|S_d^\ell|$ multiplications and G is computed one time. Finally, for one round, the number of multiplications is given by

$$\Big(\sum_{\ell=1}^{m} \Big[\Big(d(2^{d-1}-1) + d!(d-1) + 1 \Big)|S_d^\ell| + (d-1)|S_{d-1}^\ell| + (d-2)|S_{d-2}^\ell| + \ldots +$$

$$2|S_2^\ell| + |S_1^\ell|\Big]\Big) \quad (\sharp)$$

and then we have to multiply by the number of rounds R to get C.

For the $\tilde{Z}K(d)$ scheme, we have $\tilde{M} = M + n^{\lceil \frac{d}{2} \rceil}/\lceil \frac{d}{2} \rceil !$.
Then $\tilde{C} = 3\tilde{M}\tilde{R}$.

8 An Application of $ZK(3)$ and $\tilde{ZK}(3)$ to Brent Equations

8.1 Brent Equations

These equations arise when we want to compute efficiently the product of two $N \times N$ matrices in only s multiplications of entries. Of course, we require s to be less that N^3, since the naive method uses N^3 multiplications. For $N = 2$, Strassen's algorithm [19] requires 7 multiplications instead of 8 multiplications and Laderman showed that for $N = 3$, it is possible to use 23 multiplications instead of $3^3 = 27$ [9]. For $N = 2$, the least number we can obtain is seven. For $N = 3$, it is not known if 23 is the least number in the non-commutative case. For the commutative case, only 22 multiplications are needed. However we still do not know if it is the least. But the non-commutative case is more important since it allows to consider the product of matrices whose entries are themselves matrices and not only scalars. So the same technique can be generalized for larger matrices.

For sake of completeness, we recall how we get Brent equations (cf [1]). Suppose that we want to compute the product of the two matrices $A = (A_{ij})_{\substack{1 \le i \le N \\ 1 \le j \le N}}$ and $B = (B_{ij})_{\substack{1 \le i \le N \\ 1 \le j \le N}}$. We set $C = AB$ with $C = (C_{ij})_{\substack{1 \le i \le N \\ 1 \le j \le N}}$. We want to compute AB using only s multiplications. We will use the following method. We have to find coefficient $(\alpha_{abk}), (\beta_{cdk}), (\gamma_{ijk})$, with $1 \le k \le s$ and $a, b, c, d, i, j \in \{1, 2, \ldots, n\}$. These coefficients do not depend on the choice of the matrices. Now for $1 \le k \le s$, we set $L_k = \sum_{a,b} \alpha_{abk} A_{ab}$, $R_k = \sum_{cd} \beta_{cdk} B_{cd}$ and $P_k = L_k R_k$. Then we have $C_{ij} = \sum_{k=1}^{s} \gamma_{ijk} P_k$. Since matrix multiplication is a bilinear map and the space of matrices is a linear space, we can apply the process to each element of the basis and we are led to the so-called "Brent equations":

$\sum_{k=1}^{s} \gamma_{ijk} \alpha_{abk} \beta_{cdk} = \delta_{bc} \delta_{ia} \delta_{jd}$ where $a, b, c, d, i, j \in \{1, 2, \ldots, n\}$. This is a system of cubic equations and with the previous notations, we have: $n = 3sN^2$ and $m = N^6$. It is still unknown if it is possible to multiply 3×3 matrices with less than 23 multiplications. The best known algorithms so far for solving the Brent equations for the 3×3 matrix multiplication require more than 2^{80} computations. Generally the work done consists in looking for solutions in a restricted area (with more symmetries for example), or to try some algorithms such as SAT Solver for some time (as Nicolas Courtois proposed in [3]).

8.2 A Zero-Knowledge Arguments of Knowledge Based on Brent Equations

The idea is to be as close as we can of the Brent equations in order to have informal arguments to show that if somebody is able to break our authentication scheme, then there is "probably a high probability" that he will able to break the Brent equations as well. Thus we will not use exactly Brent equations to construct a zero-knowledge arguments of knowledge. In order to be very near the Brent equations, but with a solution, we have essentially made 2 changes:

a) We consider the equations mod 2. This is justified in the sense that if a solution of the Brent equation would be found mod 2, it would some insight about the real solution and allow us to progress in finding a solution (it would cut the problem in easier independent steps).
b) We changed only the constant terms in the Brent equations, i.e. all the multivariate polynomials are exactly the same, except the constant terms. Then, we have generated some random input in the equations, so our constant terms will look random. Of course, we cannot exclude the possibility that somebody may found one day an algorithm that would be able to solve the Brent equations with its very specific constant terms, and not with random constant terms. However, with all the known algorithms on multivariate polynomials this seems to be at present rather unlikely since the complexity of these known algorithms do not really depend of the specific constant terms (as soon as they are not all 0 for example). More precisely, we proceed as follows:

1. We consider the finite field $\mathbb{F}_2$.
2. We take the left part of the Brent equations with $s = 22$ in order obtain an open problem in the non-commutative case.
3. We pick randomly variables $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ in ${\mathbb{F}_2}^{N^2 s}$.
4. We deduce the values $\sum_{k=1}^{s} \gamma_{ijk}\alpha_{abk}\beta_{cdk}$ where $a, b, c, d, i, j \in \{1, 2, \ldots, n\}$. This gives the value $\boldsymbol{v} \in {\mathbb{F}_2}^M$.
5. We then use either $ZK(3)$ (or $\tilde{ZK}(3)$) to have a zero-knowledge arguments of knowledge with the system $\boldsymbol{F}(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = \boldsymbol{v}$. Here $\boldsymbol{\alpha} = (\alpha_{abk}),\ 1 \leq k \leq s,\ a, b \in \{1, 2, \ldots, n\}$, $\boldsymbol{\beta} = (\beta_{cdk}),\ 1 \leq k \leq s,\ c, d \in \{1, 2, \ldots, n\}$ and $\boldsymbol{\gamma} = (\gamma_{ijk}),\ 1 \leq k \leq s,\ i, j \in \{1, 2, \ldots, n\}$. The system is sparse and $\boldsymbol{F} \in MC(n, m, \mathbb{F}_2)$ where $n = 794$ and $m = 729$.

8.3 More Comments about Brent Equations

Solving Brent equation would be a real progress for the scientific community, and would accelerate many algorithms. Therefore we think that it is really interesting to have a security based on them or as near as possible of them. The problem is mathematically and for computer science interesting and valuable. Similarly, it is interesting to have a security based on factorization for example instead of based on a new an unknown problem, since this problem has been studied for years. Indeed, a devastating progress in factoring (as for Brent) would be scientifically interesting, i.e. it would not be just sad news for the crypto-systems, but also good news for the scientific community.

9 Morphisms of Polynomials and Systems of Cubic Equations

9.1 The IP and MP Problems

We explain the IP and MP problem in the case of quadratic forms. Let u and n two integers. On $\mathbb{F}_q$, we consider a public set $(\mathcal{A})$ of public quadratic equations:

$$(\mathcal{A}) \quad c_k = \sum_{1 \leq i \leq n,\, 1 \leq j \leq n} \gamma_{ij}^k a_i a_j + \sum_{i=1}^{n} \mu_i^k a_i + \delta_k, \quad 1 \leq k \leq u.$$

Let S be a bijective affine transformation of the variables a_i, $1 \leq i \leq n$ and T be a bijective affine transformation of the variables c_k, $1 \leq k \leq u$. We have $S(a_1, \ldots, a_n) = (x_1, \ldots, x_n)$ and $T(c_1, \ldots, c_u) = (z_1, \ldots, z_u)$. Then from $(\mathcal{A})$, we obtain another set $(\mathcal{B})$ of u equations:

$$(\mathcal{B}) \quad z_t = \sum_{1 \leq i \leq n,\, 1 \leq j \leq n} \alpha_{ij}^t x_i x_j + \sum_{i=1}^{n} \beta_i^t x_i + \omega_t, \quad 1 \leq t \leq u.$$

We say that (S, T) is an isomorphism from $(\mathcal{A})$ to $(\mathcal{B})$ and that $(\mathcal{A})$ and $(\mathcal{B})$ are isomorphic.

The IP (isomorphism of polynomials) problem is the following: if $(\mathcal{A})$ and $(\mathcal{B})$ are two public sets of u quadratic equations, find an isomorphism (S, T) from $(\mathcal{A})$ to $(\mathcal{B})$.

When S and T are not bijective the corresponding problem is the MP (morphism of polynomials) problem.

The IP problem (Isomorphism of Polynomials) has been used to construct pubic key schemes (cf [11]). On one hand, this is not a NP-complete problem since it admits an Arthur-Merlin game when the answer is yes and when the answer is no. On the other hand, the MP problem (morphisms of polynomials) where matrices are not supposed to be invertible is proved to be NP-complete [4,12] and thus is much more difficult. So it is interesting to design a public key authentication scheme based on MP. We explain briefly below how it is possible to construct such a scheme by transforming MP very efficiently into a system of equations of degree 3 and then applying our $ZK(3)$ or $\tilde{ZK}(3)$ protocols.

9.2 A Zero-Knowledge Arguments of Knowledge Based on the MP Problem

We consider the two following systems:

$$(\mathcal{A}) \quad c_k = \sum_{1 \leq i \leq n,\, 1 \leq j \leq n} \gamma_{ij}^k a_i a_j + \sum_{i=1}^{n} \mu_i^k a_i, \quad 1 \leq k \leq u$$

$$(\mathcal{B}) \quad z_t = \sum_{1 \leq i \leq p,\, 1 \leq j \leq p} \alpha_{ij}^t x_i x_j + \sum_{i=1}^{p} \beta_i^t x_i, \quad 1 \leq t \leq v$$

We want to find 2 matrices $M = (m_{rs})_{\substack{1\le r\le v\\ 1\le s\le u}}$ and $H = (h_{df})_{\substack{1\le d\le n\\ 1\le f\le p}}$ such that

$$M\begin{pmatrix} c_1 \\ \vdots \\ c_u \end{pmatrix} = \begin{pmatrix} z_1 \\ \vdots \\ z_v \end{pmatrix} \quad \text{and} \quad H\begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}.$$

For all t, $1 \le t \le v$, on one hand, we have: $z_t = \sum_{s=1}^{u} m_{ts}\Big(\sum_{\substack{1\le i\le n\\ 1\le j\le n}} \gamma_{ij}^s a_i a_j + \sum_{i=1}^{n} \mu_i^s a_i\Big)$

$z_t = \sum_{s=1}^{u} m_{ts}\Big(\sum_{\substack{1\le i\le n\\ 1\le j\le n}} \gamma_{ij}^s (\sum_{f=1}^{p} h_{if}x_f)(\sum_{b=1}^{p} h_{jb}x_b) + \sum_{i=1}^{n} \mu_i^s (\sum_{f=1}^{p} h_{if}x_f)\Big)$

$z_t = \sum_{f=1}^{p}\sum_{b=1}^{p}\Big[\sum_{s=1}^{u}\sum_{\substack{1\le i\le n\\ 1\le j\le n}} \gamma_{ij}^s m_{ts}h_{if}h_{jb}\Big]x_f x_b + \sum_{f=1}^{p}\Big[\sum_{s=1}^{u}\sum_{i=1}^{n} \mu_i^s m_{ts}h_{if}\Big]x_f$. On the other hand, we have: $z_t = \sum_{\substack{1\le i\le p\\ 1\le j\le p}} \alpha_{ij}^t x_i x_j + \sum_{i=1}^{p} \beta_i^t x_i$. This gives $\forall t\, 1 \le t \le v$, $\forall f, 1 \le f \le p$, $\forall b, 1 \le b \le p$, $\alpha_{fb}^t + \beta_f^t = \sum_{s=1}^{u}\sum_{\substack{1\le i\le n\\ 1\le j\le n}} \gamma_{ij}^s m_{ts}h_{if}h_{jb} + \sum_{s=1}^{u}\sum_{i=1}^{n} \mu_i^s m_{ts}h_{if}$. Thus we obtain a system of vp^2 cubic equations with $np + vu$ unknowns. Then it is possible to construct zero-knowledge arguments of knowledge with $ZK(3)$, (or $\tilde{ZK}(3)$).

10 Conclusion

In [15], a very efficient zero-knowledge proof based on the MQ problem (multivariate quadratic polynomials) is given. In this paper we proved that this construction can be generalized to polynomials of degree d for any $d \ge 3$ (unlike in [14] where a construction valid only for $d = 3$ was given) . We studied several constructions and we presented here the two most efficient ones denoted by $ZK(d)$ and $\tilde{ZK}(d)$. $ZK(d)$ is more efficient in terms of number of communication bits, and $\tilde{KZ}(d)$ in terms of computations. In table 4, we can compare our schemes for $d = 3$ from the recent scheme [14]. It is interesting to notice that if the polynomials are sparse, our schemes will be able to use this fact in order to be more efficient (for $\tilde{ZK}(d)$ for example, the numbers of computations is still proportional to the time to compute the polynomial on a given point). Finally, we also presented two important specific problems (Brent equations and morphisms of polynomials) that can be transformed into efficient public key schemes using $ZK(d)$ and $\tilde{ZK}(d)$.

Table 4. Comparison of schemes on 80-bit security against key-recovery attack when the impersonation probability is less than 2^{-30}

	MC	$ZK(3)$	$\tilde{ZK}(3)$
round	73	73	52
communication (bit)	53,290	74,022	560,000
arithmetic operations (times/field)	$2^{32}/\mathbb{F}_2$	$2^{33}/\mathbb{F}_2$	$2^{31}/\mathbb{F}_2$
generalization for $d \geq 4$	NO	YES	YES

References

1. Bard, G.V.: New Practical Strassen-like Approximate Matrix-multiplication Algorithms found via solving a system of cubic equations, http://www.users.math.umd.edu/~bardg/
2. Berbain, C., Gilbert, H., Patarin, J.: QUAD: A Practical Stream Cipher with Provable Security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 109–128. Springer, Heidelberg (2006)
3. Courtois, N., Bard, G.V., Hulme, D.: A new general-purpose method to multiply 3x3 matrices using only 23 multiplications. CoRR, abs/1108.2830 (2011)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completness. W.H. Freeman and Co. (1979)
5. Goldreich, O.: Foundations of Cryptography: Volume I. Basic Tools. Cambridge University Press, Cambridge (2001)
6. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. J. ACM 38, 690–728 (1991)
7. Halevi, S., Micali, S.: Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 201–215. Springer, Heidelberg (1996)
8. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced Oil and Vinegar Signature Schemes. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1492, pp. 206–222. Springer, Heidelberg (1998)
9. Laderman, J.: A noncomuutative algorithm for multiplying 3 x3 matrices using 23 multiplication. Bulletin of the American Mathematical Society 82, 126–128 (1976)
10. Matsumoto, T., Imai, H.: Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988)
11. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996)
12. Patarin, J., Goubin, L.: Trapdoor One-Way Permutations and Multivariate Polynomials. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 356–368. Springer, Heidelberg (1997)
13. Pointcheval, D.: A New Identification Scheme Based on the Perceptrons Problem. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 319–328. Springer, Heidelberg (1995)
14. Sakumoto, K.: Public-Key Identification Schemes Based on Multivariate Cubic Polynomials. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 172–189. Springer, Heidelberg (2012)

15. Sakumoto, K., Shirai, T., Hiwatari, H.: Public-Key Identification Schemes Based on Multivariate Quadratic Polynomials. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 706–723. Springer, Heidelberg (2011)
16. Shamir, A.: An Efficient Identification Scheme Based on Permuted Kernels (Extended Abstract). In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 606–609. Springer, Heidelberg (1990)
17. Stern, J.: A New Identification Scheme Based on Syndrome Decoding. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 13–21. Springer, Heidelberg (1994)
18. Stern, J.: Designing Identification Schemes with Keys of Short Size. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 164–173. Springer, Heidelberg (1994)
19. Strassen, V.: Gaussion Elimination is not optimal. Numerische Mathematik 13(3), 354–356 (1969)
20. Thomas, E.G.F.: A Polarizatin Identity for Multilinear Maps. University of Groningen - Preprint (1997)

Improved Exponentiation and Key Agreement in the Infrastructure of a Real Quadratic Field

Vanessa Dixon*, Michael J. Jacobson Jr.**, and Renate Scheidler

Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
vanessa.dixon@gmail.com, {jacobs,rscheidl}@ucalgary.ca

Abstract. We describe improvements to the performance of a key agreement protocol based in the infrastructure of a real quadratic field through investigating fast methods for exponentiating ideals. We present adaptations of non-adjacent form and signed base-3 exponentiation and compare these to the binary method. To adapt these methods, we introduce new algorithms for squaring, cubing, and dividing w-near (f, p) representations of ideals in the infrastructure. Numerical results from an implementation of the key agreement protocol using our new algorithms and all three exponentiation methods are presented, demonstrating that non-adjacent form exponentiation improves the speed of key establishment for most of the currently recommended security levels.

Keywords: real quadratic field, infrastructure, (f, p) representation, non-adjacent form exponentiation, signed base-3 exponentiation, cryptographic key agreement.

1 Introduction

In 1988, Buchmann and Williams [3] presented a key establishment protocol analogous to that of Diffie-Hellman [6], but performed in the class group of an imaginary quadratic field. Interestingly, the security of this key agreement is related to computing the class number of the field, which is known to be at least as hard as integer factorization [15, p. 360]. The following year, Buchmann and Williams proposed a method for performing an analogous key agreement protocol in the infrastructure of the principal class of a real quadratic field [4], an abelian group-like structure that was discovered by Shanks in 1972 [18]. This contribution was noteworthy because it represented the first such protocol for which the underlying structure is not a group. Furthermore, the security of this protocol is believed to be independent of the hardness assumptions used in other public-key systems, such as the difficulty of extracting discrete logarithms on algebraic curves or in finite fields. Thus, although such systems are known to sucumb to quantum algorithms, they are nevertheless a viable alternative

* The results in this paper are from the first author's M.Sc. thesis.
** The second and third authors are supported in part by NSERC of Canada.

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 214–233, 2012.

to other widely used public-key cryptosystems that can be used in the event that efficient classical algorithms are found for discrete logarithm computation or integer factorization.

The infrastructure is comprised of the set of reduced principal ideals in the maximal order of a real quadratic field, each paired with an approximation of a floating-point *distance* that keeps track of the ideal's position in the infrastructure. The challenge is to maintain sufficient accuracy throughout the key establishment protocol to make sure that both parties obtain a shared key ideal at the end. One method for handling this problem is to use a concept called w*-near* (f, p) *representations*. These were introduced in [15], based on ideas originally presented in [13].

In this paper, we consider three methods for exponentiation of w-near (f, p) representations: binary exponentiation (BINEXP), non-adjacent form exponentiation (NAFEXP), and a signed base-3 exponentiation (SB3EXP). The first of these was previously investigated in [15], while the other two are new. The non-adjacent form of an integer is sparser than its binary form, so NAFEXP requires fewer multiplication/division steps than BINEXP on average. The base-3 representation of an integer is shorter than its binary form, so SB3EXP requires fewer cubings than BINEXP requires squarings. Hence, if cubing and division have similar time requirements to squaring and multiplying, we would expected that these algorithms improve the speed at which exponentiation can be done.

A major ingredient of all our exponentiation techniques is the efficient multiplication method for w-near (f, p) representations given as Algorithm 11.3 on pp. 275-276 of [15]. Algorithms for squaring, cubing, and division of (f, p)-representations are also required for NAFEXP and SB3EXP and are newly introduced herein. A comparative precision analysis is provided for each of these new algorithms and for key agreement using each of our three exponentiation methods. All relevant algorithms were implemented in C using the GNU Multiprecision Arithmetic Library, and test trials were run to examine their efficiency. Our timing results show that NAFEXP mostly outperforms BINEXP, whereas SB3EXP does not.

2 Infrastructure of a Real Quadratic Field

For a general introduction to real quadratic fields and their infrastructure, the reader is referred to [15]. Throughout this paper, we fix a positive square-free integer $D > 1$, and set $r = 2$ if $D \equiv 1 \pmod 4$, $r = 1$ otherwise. The field and 2-dimensional $\mathbb{Q}$-vector space $\mathbb{K} = \mathbb{Q} \oplus \mathbb{Q}\sqrt{D}$ is a *real quadratic field*. Its *discriminant* is $\Delta = 4D/r^2$, and its *maximal order* is the subring and rank 2 $\mathbb{Z}$-module $\mathcal{O} = \mathbb{Z} \oplus \mathbb{Z}\omega$, where $\omega = (r - 1 + \sqrt{D})/r$.

2.1 Ideals and Infrastructure

An $\mathcal{O}$*-ideal* (or *ideal* for short) is an additive subgroup of $\mathcal{O}$ that is closed under multiplication by elements in $\mathcal{O}$. An $\mathcal{O}$-ideal $\mathfrak{a}$ is *principal* if it consists of all the

$\mathcal{O}$-multiples of some element $\alpha \in \mathcal{O}$, called a *generator* of $\mathfrak{a}$; we write $\mathfrak{a} = (\alpha)$. It will at times be useful to consider *fractional* principal ideals $\mathfrak{a} = (\alpha)$ with $\alpha \in \mathbb{K}$, which are no longer subsets of $\mathcal{O}$, but $(d)\mathfrak{a} \subseteq \mathcal{O}$ for some $d \in \mathbb{Z}$. Two non-zero $\mathcal{O}$-ideals $\mathfrak{a}$ and $\mathfrak{b}$ are *equivalent* is there exists a non-zero $\theta \in \mathbb{K}$ with $\mathfrak{b} = (\theta)\mathfrak{a}$. Note that the non-zero principal $\mathcal{O}$-ideals are exactly the $\mathcal{O}$-ideals that are equivalent to $\mathcal{O}$.

The non-zero $\mathcal{O}$-ideals are exactly the rank 2 $\mathbb{Z}$-submodules of $\mathcal{O}$ of the form $\mathfrak{a} = \mathbb{Z}\,SQ/r \oplus \mathbb{Z}\,S(P + \sqrt{D})/r$ with $S, Q, P \in \mathbb{Z}$, $S, Q > 0$, r dividing Q and rQ dividing $D - P^2$. Here, S and Q are unique and P is unique modulo Q. The *norm* of $\mathfrak{a}$ is the positive integer $N(\mathfrak{a}) = S^2Q$. An $\mathcal{O}$-ideal $\mathfrak{a}$ is *primitive* if $S = 1$, in which case we simply write $\mathfrak{a} = (Q, P)$. The *conjugate* of a primitive ideal $\mathfrak{a} = (Q, P)$ is the primitive ideal $\overline{\mathfrak{a}} = (Q, -P)$; note that $\mathfrak{a}\overline{\mathfrak{a}} = (N(\mathfrak{a})) = (Q)$.

A primitive ideal $\mathfrak{a} = (Q, P)$ is *reduced* if P can be taken modulo Q so that $0 < \sqrt{D} - P < Q < \sqrt{D} + P$. This forces $0 < P < \sqrt{D}$ and $0 < Q < 2\sqrt{D}$, so P and Q are bounded. Consequently, the number of reduced $\mathcal{O}$-ideals is finite.

The *infrastructure* of $\mathbb{K}$ is the set $\mathcal{R}$ of all reduced principal $\mathcal{O}$-ideals. Each infrastructure ideal $\mathfrak{a}$ is associated with its unique *distance* $\delta(\mathfrak{a}) = \log \alpha \in \mathbb{R}^{\geq 0}$, where $\alpha \in \mathcal{O}$ is the smallest generator of $\mathfrak{a}$ that is at least 1. The infrastructure is thus a finite set that is ordered by distance, where the first ideal is $\mathcal{O}$ and has distance 0. *Baby steps* move cyclically through the infrastructure, obtaining from any infrastructure ideal $\mathfrak{a}$ the next infrastructure ideal $\rho(\mathfrak{a})$ in the distance ordering, along with the *relative distance* $\delta(\rho(\mathfrak{a})) - \delta(\mathfrak{a}) \in \mathbb{R}^{>0}$.

The product $\mathfrak{ab}$ of two reduced principal ideals $\mathfrak{a}, \mathfrak{b}$ is generally a non-reduced (and even non-primitive) principal $\mathcal{O}$-ideal; more exactly, $\mathfrak{ab} = (S)\mathfrak{c}$ with $S \in \mathbb{Z}^+$ and $\mathfrak{c}$ a primitive $\mathcal{O}$-ideal. *Reduction* applies the same arithmetic as the baby step operation to $\mathfrak{c}$, producing a reduced ideal equivalent to $\mathfrak{ab}$. Suppose $\mathfrak{a}$ and $\mathfrak{b}$ are infrastructure ideals, and let $\mathfrak{r} = (\theta)\mathfrak{ab} \in \mathcal{R}$, with $\theta \in \mathbb{K}$, be the first reduced ideal thus obtained. Then the operation that computes $\mathfrak{r}$ and θ from $\mathfrak{a}$ and $\mathfrak{b}$ is a *giant step*. Instead of using multiplication with subsequent reduction, giant steps can be performed more efficiently using the NUCOMP algorithm first proposed by Shanks [19]. The distance "error" $\log \theta = \delta(\mathfrak{r}) - \delta(\mathfrak{a}) - \delta(\mathfrak{b}) \in \mathbb{R}$ is generally very small compared to the distances of $\mathfrak{a}$ and $\mathfrak{b}$. It follows that under the giant step operation, the infrastructure behaves almost like a finite abelian group, where the identity is $\mathcal{O}$ and the "inverse" operation is conjugation; associativity fails, but only barely since the distance is nearly additive under giant steps.

2.2 (f, p) Representations

When performing infrastructure arithmetic, one needs to keep track of the relative distances of ideals which are real numbers. These are approximated using *w-near* (f, p) *representations* of ideals, a concept first introduced in [13] and subsequently developed in [14].

Definition 1. *[15, Definition 11.1, p. 267 and Section 11.2, p. 270] Let* $p \in \mathbb{Z}^+$, $f \in \mathbb{R}^{\geq 1}$ *and* $\mathfrak{a}$ *an* $\mathcal{O}$*-ideal. An* (f, p) representation *of* $\mathfrak{a}$ *is a triple of parameters* $(\mathfrak{b}, d, k)$ *where*

1. $\mathfrak{b}$ *is an* $\mathcal{O}$*-ideal equivalent to* $\mathfrak{a}$*,* $d \in \mathbb{Z}^+$ *with* $2^p < d \leq 2^{p+1}$*,* $k \in \mathbb{Z}$*, and*
2. *there exists* $\theta \in \mathbb{K}$ *such that* $\mathfrak{b} = (\theta)\mathfrak{a}$ *with* $|2^{p-k}\theta/d - 1| < f/2^p$*.*

The (f, p) *representation* $(\mathfrak{b}, d, k)$ *is* reduced *if* $\mathfrak{b}$ *is a reduced* $\mathcal{O}$*-ideal, and is* w-near *if in addition*

3. $k < w$ *and*
4. *if* $\rho(\mathfrak{b}) = (\phi)\mathfrak{b}$*, then there exist* $k', d' \in \mathbb{Z}$ *with* $k' \geq w$ *and* $2^p < d' \leq 2^{p+1}$ *such that* $|2^{p-k'}\theta\phi/d' - 1| < f/2^p$*.*

The intuition behind (f, p)-representations is that $d2^{k-p}$ is an approximation of the (generally unknown) relative generator θ of $\mathfrak{b}$ with respect to the (generally unknown) ideal $\mathfrak{a}$ with an accuracy of $f2^{-p}$. In this respect, p can be regarded as the *precision* of the approximation which will be fixed throughout our computations, and f as a measure of the error which will increase with each computation (error propagation). Also, since $2^p < d \leq 2^{p+1}$ and $d2^{k-p} \approx \theta$, k represents a rough approximation of the relative distance $\log \theta$ from $\mathfrak{b}$ to $\mathfrak{a}$.

Property 4 of Definition 1 implies that if $(\mathfrak{b}, d, k)$ is a w-near (f, p) representation of $\mathfrak{a}$, then $(\rho(\mathfrak{b}), d', k')$ is a reduced (but not necessarily w-near) (f, p) representation of $\mathfrak{a}$. In this case, the bounds in Lemma 11.3, p. 270, of [15] imply that qualitatively, θ and k are approximated by 2^w and w, respectively. Moreover, if $(\mathfrak{c}, g, h)$ is another w-near (f, p) representation of $\mathfrak{a}$, then by [15, Theorem 11.4, p. 271], $\mathfrak{b} \in \{\rho^{-2}(\mathfrak{c}), \rho^{-1}(\mathfrak{c}), \mathfrak{c}, \rho(\mathfrak{c}), \rho^2(\mathfrak{c})\}$. Hence, any two w-near (f, p) representations of the same ideal are within a few baby steps of one another. When exponentiation of w-near (f, p) representations is used in cryptographic key agreement, the preservation of the w-near property throughout the exponentiation algorithm will thus guarantee that the keys computed by Alice and Bob will be close to each other in the infrastructure. Experimentally it has been verified that if f is much less than 2^p, it is most often the case that $\mathfrak{b} = \mathfrak{c}$ (see also [15, Theorem 11.5, p. 272]).

Using w-near (f, p) representations also leads to computational improvements when performing infrastructure arithmetic. Each giant step $\mathfrak{r} = (\theta)\mathfrak{a}\mathfrak{b}$ performed on infrastructure ideals $\mathfrak{a}$ and $\mathfrak{b}$ produces a distance error $\log \theta$ which is generally negative. Qualitatively, this means that a small distance shortfall of $\log \theta$ is introduced in each giant step. By calculating the average distance lost due to reduction following ideal multiplication, the number of operations required to account for this lost distance can be decreased. One purpose of using w-near (f, p) representations is to counter this "head wind": the approximations of the distances of $\mathfrak{a}$ and $\mathfrak{b}$ after a giant step are adjusted (increased) so that the distance lost during the giant step is added back, thereby reducing the computational overhead required when exponentiation and key agreement are performed.

We now provide a summary of known and new results on basic arithmetic involving (f, p) representations. Our first theorem gives parameters for the product of two such representations.

Theorem 1. [15, Theorem 11.2, p. 268] *Let* $(\mathfrak{b}', d', k')$ *be an* (f', p) *representation of an* $\mathcal{O}$*-ideal* $\mathfrak{a}'$ *and* $(\mathfrak{b}'', d'', k'')$ *an* (f'', p) *representation of an* $\mathcal{O}$*-ideal* $\mathfrak{a}''$*.*

If $d'd'' \leq 2^{2p+1}$, *put* $e = \lceil d'd''/2^p \rceil$ *and* $h = k' + k''$, *else put* $e = \lceil d'd''/2^{p+1} \rceil$ *and* $h = k' + k'' + 1$. *Then* $(\mathfrak{b}'\mathfrak{b}'', e, h)$ *is an* (f, p) *representation of* $\mathfrak{a}'\mathfrak{a}''$ *where* $f = f' + f'' + 2^{-p} f' f'' + 1$.

Given an (f, p) representation of some ideal, we can compute a w-near $(f + 9/8, p)$ representation of the same ideal using an algorithm called WNEAR. The complete algorithm can be found in [15, Algorithm 11.2, pp. 454-456].

Multiplication of two w-near representations, obtaining a w-near representation of the product, is achieved using the WMULT algorithm [15, Algorithm 11.3, pp. 275-276]. WMULT first computes the ideal product followed by some reduction steps along with the necessary parameters for a reduced representation of that product. This increases the f-value of Theorem 1 by $9/8$; see [15, Algorithm 11.1, p. 269]. Now the w-near property is re-established for the resulting representation of the product using WNEAR, which increases the f-value by another $9/8$. Hence, given (f', p) and (f'', p) representations $(\mathfrak{b}', d', k')$ and $(\mathfrak{b}'', d'', k'')$ of $\mathfrak{a}'$ and $\mathfrak{a}''$, respectively, WMULT computes a w-near (f, p) representation of $\mathfrak{a}'\mathfrak{a}''$, where $f = f' + f'' + 2^{-p} f' f'' + 13/4$. For the complete algorithm, see [15, Algorithm 11.3, pp. 275-276].

Squaring is the special case of multiplication applied to two identical inputs and requires $\mathfrak{a}' = \mathfrak{a}''$, $\mathfrak{b}' = \mathfrak{b}''$, $d' = d''$ and $k' = k''$ in Theorem 1. The corresponding algorithm, WDUPL, is presented in the Appendix as Algorithm A.3. It contains simplifications over the general WMULT algorithm which achieve some computational improvements when performing exponentiation on (f, p) representations.

3 Cubing and Division with (f, p) Representations

An integer expressed in terms of a larger base has fewer terms compared to a smaller base, thus requiring fewer steps when used as an exponent in exponentiation. For example, it might be desirable to represent the exponent in base-3. This type of exponentiation in turn requires a cubing method for (f, p) representations and, in case signed digits are allowed, a division technique for (f, p) representations.

An algorithm for cubing ideals can be found in [11, Algorithm 4, p. 16]. This technique outputs an ideal that is at most two steps away from being reduced. The following theorem gives the parameters of an (f, p) representation $((\mathfrak{b}')^3, e, h)$ of $(\mathfrak{a}')^3$, given an (f', p) representation of $\mathfrak{a}'$. This result is attributed to A. Silvester; a proof will appear in his doctoral dissertation [20].

Theorem 2. *Let* $(\mathfrak{b}', d', k')$ *be an* (f', p) *representation of an* $\mathcal{O}$*-ideal* $\mathfrak{a}'$. *If* $(d')^3 \leq 2^{3p+1}$, *put* $e = \lceil (d')^3/2^{2p} \rceil$ *and* $h = 3k'$. *If* $2^{3p+1} < (d')^3 \leq 2^{3p+2}$, *put* $e = \lceil d'^3/2^{2p+1} \rceil$ *and* $h = 3k + 1$. *If* $(d')^3 > 2^{3p+2}$, *put* $e = \lceil (d')^3/2^{2p+2} \rceil$ *and* $h = 3k' + 2$. *Then* $((\mathfrak{b}')^3, e, h)$ *is an* (f, p) *representation of* $(\mathfrak{a}')^3$, *where* $f = 3f' + 2^{-p} 3(f')^2 + 2^{-2p}(f')^3 + 1$.

Obtaining a w-near representation of a cube proceeds analogous to multiplication or squaring: first compute the ideal cube, then apply reduction and compute the

parameters of the corresponding reduced representation, then restore the w-near property and again compute the parameters of the resulting w-near representation. The overall algorithm, called WCUBE, is presented as Algorithm A.4 in the Appendix. Note that some minor mistakes from [11, Algorithm 4, p. 16] have been corrected.[1] As before, given an (f', p) representation $(\mathfrak{b}', d', k')$ of an $\mathcal{O}$-ideal $\mathfrak{a}'$, WCUBE computes a w-near (f, p) representation of $(\mathfrak{a}')^3$, where $f = 3f' + 2^{-p}3(f')^2 + 2^{-2p}(f')^3 + 13/4$.

In order to implement exponentiation using signed digits such as NAF or signed base-3 exponentiation, we require a *division* algorithm for (f, p) representations. First, we give a brief overview of how ideal division is accomplished. Let $\mathfrak{b}' = (\theta')\mathfrak{a}'$ and $\mathfrak{b}'' = (\theta'')\mathfrak{a}''$ with $\mathcal{O}$-ideals $\mathfrak{a}', \mathfrak{a}''$, $\mathfrak{b}'$, $\mathfrak{b}''$ and relative generators $\theta', \theta'' \in \mathbb{K}$. Then $\mathfrak{b}'\overline{\mathfrak{b}''}(\theta'')\mathfrak{a}'' = \mathfrak{b}'\overline{\mathfrak{b}''}\mathfrak{b}'' = \mathfrak{b}'(N(\mathfrak{b}'')) = (\theta')\mathfrak{a}'(N(\mathfrak{b}''))$. Suppose that $\mathfrak{a}''$ divides $\mathfrak{a}'$, i.e. there exists an $\mathcal{O}$-ideal $\mathfrak{c}$ such that $\mathfrak{a}' = \mathfrak{c}\mathfrak{a}''$. For example, in our context, we will always have $\mathfrak{a}' = (\mathfrak{a}'')^n$ for some $n \in \mathbb{Z}^+$. Then $\mathfrak{b}'\overline{\mathfrak{b}''} = (\theta' N(\mathfrak{b}'')/\theta'')\mathfrak{c}$. Thus, when "dividing" an (f', p) representation $(\mathfrak{b}', d', k')$ of $\mathfrak{a}'$ by an (f'', p) representation $(\mathfrak{b}'', d'', k'')$ of $\mathfrak{a}''$, the resulting (f, p) representation should approximate the new relative generator $\theta = \theta' N(\mathfrak{b}'')/\theta''$ of $\mathfrak{b}'\overline{\mathfrak{b}''}$ with respect to $\mathfrak{c}$. The following theorem provides the exact parameters. Its proof is rather long and detailed, so for the sake of brevity, we only state the result here; the complete proof can be found in [7, Theorem 3.6.2].

Theorem 3. [7, Theorem 3.6.2, p. 51] *Let $(\mathfrak{b}', d', k')$ be an (f', p) representation of an $\mathcal{O}$-ideal $\mathfrak{a}'$ and $(\mathfrak{b}'', d'', k'')$ an (f'', p) representation of an $\mathcal{O}$-ideal $\mathfrak{a}''$ dividing $\mathfrak{a}'$, with $\mathfrak{a}' = \mathfrak{c}\mathfrak{a}''$ for some $\mathcal{O}$-ideal $\mathfrak{c}$. Define $\kappa \in \mathbb{Z}$ via $2^\kappa < N(\mathfrak{b}'') \leq 2^{\kappa+1}$ and $d^* = d'N(\mathfrak{b}'')/(2^\kappa d'')$. If $1/2 < d^* \leq 1$, put $e = \lceil 2^{p+1}d^* \rceil$ and $h = k' - k'' + \kappa - 1$. If $1 < d^* \leq 2$, put $e = \lceil 2^p d^* \rceil$ and $h = k' - k'' + \kappa$. If $2 < d^* < 4$, put $e = \lceil 2^{p-1}d^* \rceil$ and $h = k' - k'' + \kappa + 1$. Then $(\mathfrak{b}'\overline{\mathfrak{b}''}, e, h)$ is an (f, p) representation of $\mathfrak{c}$ where $f = f' + f''/(1 - f''/2^p) + f'f''/(2^p - f'') + 1$.*

WDIV performs division on w-near representations, followed by WNEAR. It takes as input a w-near (f', p) representation $(\mathfrak{b}', d', k')$ of $\mathfrak{a}'$ and a w-near (f'', p) representation $(\mathfrak{b}'', d'', k'')$ of $\mathfrak{a}''$ where $\mathfrak{a}''$ divides $\mathfrak{a}'$, and computes a w-near (f, p) representation of the quotient ideal, with $f = f' + f''/(1 - f''/2^p) + f'f''/(2^p - f'') + 13/4$. It is presented in the Appendix as Algorithm A.5.

4 Non-adjacent Form and Signed Base-3 Exponentiation

To explore how to improve the performance of key agreement in the infrastructure of a real quadratic field, we examine various exponentiation algorithms and adapt them to the setting of w-near (f, p) representations. The well known method of binary exponentiation (BINEXP) was already presented in [15]; it is

[1] The algorithm as presented in [11] computes $N = Q'/S$, $L = NQ'/r^2$, $K = R_0 v_1(2 + v_1(v_1(Q_0/r)(R_0/r)(2P_0)/r))$ (mod L) on line 3 and $M_2 = (R_i(P' + P'') + R'S)/L$ on line 19. The corrections herein are attributed to M. Jacobson, A. Silvester, and V. Dixon.

given for the sake of completeness as Algorithm A.6 in the Appendix. Here, we investigate non-adjacent form exponentiation (NAFEXP) and a signed base-3 exponentiation method (SB3EXP) for w-near (f,p) representations.

The *non-adjacent form* (NAF) of an integer is the signed base-2 representation for which no two adjacent digits are both non-zero. Given a positive integer n, the non-adjacent form of n is $n = \sum_{i=0}^{\ell_N} n_i 2^{\ell_N - i}$ where $n_0 = 1$, $n_i \in \{-1, 0, 1\}$, and $n_i n_{i-1} = 0$ for $1 \le i \le \ell_N$. The NAF of an integer can be computed using for example [10, Algorithm 3.30, p. 98].

The NAF has several useful properties that make it amenable for exponentiation; see [10, Theorem 3.29, p. 98] for example. The average number of required operations decreases from $(3/2)\log n$ to $(4/3)\log n$ as compared to BINEXP. However, because the binary digits are signed, both multiplication and division steps are required. If our division algorithm WDIV is approximately as fast as the multiplication algorithm WMULT, then we expect an improvement.

The algorithm NAFEXP, using non-adjacent form to exponentiate w-near (f,p) representations of $\mathcal{O}$-ideals, is given in the Appendix as Algorithm A.7. It is an adaptation of [10, Algorithm 3.31, p. 99], a binary NAF method for point multiplication on elliptic curves, adapted to work in the infrastructure of a real quadratic field. Theorem 3.29 of [10] implies that, compared to BINEXP, NAFEXP requires at most one more squaring, but reduces the average number of multiplications from $\ell_B/2$ to $\ell_N/3 \le (\ell_B+1)/3$ multiplications and divisions; here $\ell_B + 1$ is the binary length.

In order to use NAFEXP for key establishment, we must determine bounds on the error estimate f in the final w-near (f,p) representation of an ideal $\mathfrak{a}^n$. The proof is again long and very technical, so we only quote the final result. It establishes an upper bound a_i, given recursively in terms of a_{i-1}, on the value $f = f_i$ after the i-th step of the exponentiation. Solving the recurrence for a_i provides an upper bound a_{ℓ_N} on the final value $f = f_{\ell_N}$. The complete proof can be found in [7, Section 4.2.1].

Theorem 4. [7, Theorem 4.2.4, p. 73] *Let $p \ge 8$, $n \ge 2$, $h \ge \max\{16, \log_2 n\}$ and $f_0 < 2^{p-4}$. Put $m = 3.54 f_0 + 10.72$. If $hmn < 2^p$, then after NAFEXP has executed on input an (f_0, p) representation and an exponent n, the resulting (f,p) representation satisfies $f < mn$, and hence $f < 2^p/h \le 2^{p-4}$.*

Both the binary form and the non-adjacent form of an integer are base-2 representations. A base-3 representation could be advantageous because it has a shorter length. However, cubing an (f,p) representation is a more costly operation than squaring. This led us to investigate if using a signed base-3 exponentiation would provide advantages over the binary method of exponentiation.

The signed base-3 representation of an integer n is $n = \sum_{i=0}^{\ell_S} n_i 3^{\ell_S - i}$, where $n_0 = 1$ and $n_i \in \{-1, 0, 1\}$ for $1 \le i \le \ell_S$. This representation can be computed by repeatedly dividing by 3 and choosing the remainders in $\{-1, 0, 1\}$. The premier advantage of this representation is its shorter length compared to the NAF or binary representation, namely $\ell_S \le \log_3 n + 1 = \log_2 n / \log_2 3 + 1$. However, unlike NAF, it is not generally true that no two adjacent digits are non-zero. Since every digit in the signed base-3 representation can be either $0, 1$

or -1 with an expected uniform distribution, the average density of non-zero bits is $2/3$. So the number of multiplication and division operations is on average

$$2\ell_S/3 \leq (2\log_3 n + 1)/3 = (2\log_2 n)/(3\log_2 3) + 1/3 \approx 0.42\log_2 n + 0.33.$$

The average number of operations for SB3EXP is thus less than that required for BINEXP, but greater than that required for NAFEXP.

SB3EXP employs a signed base-3 representation of the exponent in a cube and multiply/divide method for exponentiating w-near (f, p) representations, using the algorithms WCUBE, WMULT and WDIV from the previous two sections. The algorithm is given in the Appendix as Algorithm A.8. The precision analysis and proof proceed analogously to that of NAFEXP. Again, we only quote the final result; the complete proof can be found in [7, Section 4.3.1].

Theorem 5. [7, Theorem 4.3.3, p. 84] *Let $p \geq 8$, $n \geq 2$, $h \geq \max\{16, \log_2 n\}$ and $f_0 < 2^{p-4}$. Put $m = 13.7f_0 + 41.3$. If $hmn < 2^p$, then after SB3EXP has executed on input an (f_0, p) representation and an exponent n, the resulting w-near (f, p) representation satisfies $f < mn$, and hence $f < 2^p/h \leq 2^{p-4}$.*

5 Key Agreement Protocols

NAFEXP and SB3EXP can be used in a Diffie-Hellman type key agreement protocol in which two parties, Alice and Bob, establish a shared secret cryptographic key suitable for use in a block cipher such as AES. The protocol is based on the key agreement procedure presented by Buchmann and Williams [4]. Alice and Bob agree on a real quadratic field $\mathbb{K}$, an ideal $\mathfrak{g}$ in the infrastructure of $\mathbb{K}$, and an exponent bound B. Informally, the protocol proceeds as follows.

Protocol 6. (Infrastructure cryptographic key agreement [15, p. 365]**)**

1. Alice secretly generates a random integer a with $0 < a < B$. She computes an infrastructure ideal $\mathfrak{a} = (\theta_a)\mathfrak{g}^a$ with $\theta_a \approx 1$ and sends $\mathfrak{a}$ to Bob.
   Bob secretly generates a random integer b with $0 < b < B$. He computes an infrastructure ideal $\mathfrak{b} = (\theta_b)\mathfrak{g}^b$ with $\theta_b \approx 1$ and sends $\mathfrak{b}$ to Alice.
2. Alice computes $\mathfrak{k}_a = (\theta_\alpha)\mathfrak{b}^a$, where $\theta_\alpha \approx 1$.
   Bob computes $\mathfrak{k}_b = (\theta_\beta)\mathfrak{a}^b$, where $\theta_\beta \approx 1$.

Clearly, $\mathfrak{k}_a$ and $\mathfrak{k}_b$ are both equivalent to $\mathfrak{g}^{ab}$. Ensuring that the relative generators in each step are close to 1 guarantees that $\mathfrak{k}_a = (\alpha)\mathfrak{g}^{ab}$ and $\mathfrak{k}_b = (\beta)\mathfrak{g}^{ab}$ where $\alpha, \beta \approx 1$. By tracking the relative generators with enough precision, it is possible to ensure that $\mathfrak{k}_a = \mathfrak{k}_b$. However, if the precision requirements are relaxed, then the computation speed and memory requirements of the protocol may be improved. In this case, the key ideal $\mathfrak{k}_a$ computed by Alice may not be the same as Bob's key ideal $\mathfrak{k}_b$, but will instead be within a few baby steps of $\mathfrak{k}_b$, resulting in a small set of possible key ideals. The ambiguity arising from this can be resolved by encrypting and decrypting a message [15, p. 369].

Approximating the relative generators can be accomplished using w-near (f,p) representations. We adapt the key agreement protocol from [15, Protocol 14.1, pp. 368-369] to use the exponentiation algorithms NAFEXP and SB3EXP from the previous section. In this protocol, Alice and Bob agree on a discriminant Δ, a w-near (f,p) representation $(\mathfrak{g}_0, d_0, k_0)$ of some reduced principal ideal in the infrastructure of $\mathbb{Q}(\sqrt{\Delta})$, an exponent bound B and a precision value p which depends on the exponentiation algorithm used and must be chosen large enough so that Alice's and Bob's keys are close to each other in the infrastructure (see Theorem 8 below). As described in [7, p. 88], the value $w = \lceil (\log \Delta)/4 \rceil$ is selected in order to minimize the number of required adjustment steps in the infrastructure. We also use the fact that by [15, Lemma 14.2, p. 367], if $(\mathfrak{b}, d, k)$ is a w-near (f,p) representation of some $\mathcal{O}$-ideal and $r < p$, then $(\mathfrak{b}, d', k)$ is a w-near $(f + 2^r, p)$ representation of the same $\mathcal{O}$-ideal with $d' = 2^r \lceil 2^{-r} d \rceil$. This means that d-values can be truncated by r bits at the expense of an error increase of 2^r. In the protocol, EXP is one of BINEXP, NAFEXP or SB3EXP, and $r = \lfloor \log_2 B \rfloor$.

Protocol 7. (Key agreement using (f,p) representations)

1. Alice secretly generates a random integer a with $0 < a < B$, computes $(\mathfrak{a}, d_a, k_a) = \mathrm{EXP}((\mathfrak{g}_0, d_0, k_0), a, w, p)$, and sends $(\mathfrak{a}, \lceil 2^{-r} d_a \rceil, k_a)$ to Bob.
   Bob secretly generates a random integer b with $0 < b < B$, computes $(\mathfrak{b}, d_b, k_b) = \mathrm{EXP}((\mathfrak{g}_0, d_0, k_0), b, w, p)$, and sends $(\mathfrak{b}, \lceil 2^{-r} d_b \rceil, k_b)$ to Alice.
2. Alice computes $(\mathfrak{k}, d, k) = \mathrm{EXP}((\mathfrak{a}, 2^r \lceil 2^{-r} d_a \rceil, k_a), b, w, p)$.
   Bob computes $(\mathfrak{m}, e, h) = \mathrm{EXP}((\mathfrak{b}, 2^r \lceil 2^{-r} d_b \rceil, k_b), a, w, p)$.

If all parameters are chosen appropriately, then we generally expect that $\mathfrak{k} = \mathfrak{m}$, which is the shared key ideal. In general, by Theorem 8 below, $\mathfrak{k}$ is within two baby steps of $\mathfrak{m}$ in either direction, and a common key ideal can be established through a test encryption/decryption as mentioned above.

The security of Protocol 7 rests on the assumption that the principal ideal problem is hard. Given an $\mathcal{O}$-ideal $\mathfrak{a}$, the *principal ideal problem* is to determine whether $\mathfrak{a}$ is principal and, if so, compute an approximation of $\log \alpha$ where $\mathfrak{a} = (\alpha)$; see [15, Definition 13.21, p. 331]. Using an algorithm described in [15, Section 13.5, pp. 331-333], it has been established [15, Theorem 13.24, p. 332] that assuming the Generalized Riemann Hypothesis and the Extended Riemann Hypothesis, the principal ideal problem in a real quadratic field of discriminant $\Delta \geq 42$ can be solved in expected time

$$L_\Delta[1/2, \sqrt{2} + o(1)] = \exp\left((\sqrt{2} + o(1))(\log|\Delta|)^{1/2}(\log\log|\Delta|)^{1/2}\right).$$

6 Error Analysis for Key Agreement

In order for Protocol 7 to be successful, the precision p must be sufficiently high to ensure that Alice and Bob's keys are within two baby steps (backwards or forwards) of each other. For BINEXP, this was analyzed in [15], while the corresponding results for NAFEXP and SB3EXP are new. The proofs are very technical and are thus omitted herein.

Theorem 8. *Let $p, B \in \mathbb{Z}^+$ with $B \geq 14$ and set $r = \lfloor \log_2 B \rfloor$. Let $a, b, \in \mathbb{Z}$ with $0 < a, b < B$, and let $(\mathfrak{g}_0, d_0, k_0)$ be a $\lceil (\log \Delta)/4 \rceil$-near $(17/8, p)$ representation of a reduced principal ideal $\mathfrak{g}$. Set*

$$C = \begin{cases} 66 & \text{if EXP} = \text{BINEXP}, \\ 68 & \text{if EXP} = \text{NAFEXP}, \\ 982 & \text{if EXP} = \text{SB3EXP}. \end{cases}$$

If $2^p \geq CB^2 \max\{16, \log_2 B\}$, then $(\mathfrak{k}, d, k)$ and $(\mathfrak{m}, e, h)$ as given in round 2 of Protocol 7 are w-near (f, p) representations of $\mathfrak{g}^{ab}$ with $f < 2^{p-4}$. Hence $\mathfrak{k} \in \{\rho^{-2}(\mathfrak{m}), \rho^{-1}(\mathfrak{m}), \mathfrak{m}, \rho(\mathfrak{m}), \rho^2(\mathfrak{m})\}$.

Proof. For BINEXP, see [15, Theorem 14.3, p. 367, and Theorem 14.4, p. 368]. For NAFEXP, see [7, Theorems 5.3.1, p. 93, and Theorem 5.3.2, p. 94]. Finally, for SB3EXP, see [7, Theorem 5.4.1, p. 96, and Theorem 5.4.2, p. 97].

Theorem 8 can be used directly for key agreement: Alice and Bob simply use a precision value p such that $p \geq \log_2(CB^2 \max\{16, \log_2 B\})$ where C is the appropriate value as specified in the theorem. Note that the above bounds on p imply that NAFEXP requires at most one more bit of precision than BINEXP, and SB3EXP at most 15 bits more, to guarantee the result of Theorem 8.

7 Numerical Results

The algorithms BINEXP, NAFEXP and SB3EXP were implemented in C on a Dell Power Edge R910 server provided by the Department of Mathematics and Statistics at the University of Calgary. This server has 64 logical CPUs Intel(R) Xeon(R) CPU X7550 @ 2.00GHz with 128G RAM. The operating system is Red Hat Enterprise Linux Server 5.6. The GNU Multiple Precision Arithmetic Library (GMP) [8] was used for integer arithmetic.

We used discriminants with the bit lengths recommended in [2], which provide the same level of security as block ciphers with 112, 128, 192, and 256 bit keys as recommended by NIST [1]. We also used the exponent bound $B = 2^{2k}$ where k is the number of bits in the corresponding block cipher (112, 128, 192, 256) [15, pp. 372-373]. This bound ensures that an attack on the protocol using a baby-step giant-step method would take time approximately 2^k, which is roughly the same time required to solve the principal ideal problem (which is believed to be hard) using index calculus [15, p. 372]. The values of p and w were computed as described in the previous section. The parameters used in Protocol 7 that were not trial dependent are listed in Table 1.

The choice of the discriminant Δ of $\mathbb{K}$ is important for cryptographic security. Heuristically, to ensure that the infrastructure has a cardinality of order $\sqrt{\Delta}$, the best choices are prime discriminants $\Delta \equiv 1 \pmod 4$; see the discussion on p. 371 of [15]. Hence, for each trial, we generated a random probable prime value $\Delta = D \equiv 1 \pmod 4$ using random number generating and probable prime finding algorithms in GMP.

Table 1. Parameters for the trials. The values of p_B, p_N, p_S correspond to the precision values of BINEXP, NAFEXP, and SB3EXP, respectively.

$\log_2 B$	$\log_2 \Delta$	w	p_B	p_N	p_S
224	1341	335	462	462	446
256	1818	454	527	527	530
384	3586	896	783	783	787
512	5957	1489	1040	1040	1043

For better performance, the partial GCD steps (lines 7-12 of WDUPL and lines 11-16 of WCUBE) were implemented using Lehmer's algorithm [16,17]; this is not reflected in the pseudocode as it appears in the Appendix .

The entries in Table 2 are the result of 10 000 trials for each discriminant size. We used a w-near $(17/8, p)$ representation of $\rho^5(\mathcal{O})$, obtained by running WNEAR on input the reduced $(1, p)$ representation $(\rho^5(\mathcal{O}), 2^p + 1, 0)$ of $\rho^5(\mathcal{O})$, as the input ideal in each trial. Two exponents $a, b < B$ were also generated randomly for each trial. The time required to perform Protocol 7 (double exponentiation) was recorded, and we took the average over all trials. Columns 3, 4 and 6 record these times using BINEXP, NAFEXP and SB3EXP, respectively. Columns 5 and 7 show the percentage difference between the new methods (NAFEXP and SB3EXP) and BINEXP, computed using the formula $100\% \cdot (t_B - t)/t_B$ where t_B is the average CPU time from column 3 of Table 2 and t is the average CPU time from column 4 for NAFEXP or column 6 for SB3EXP from the same table. We see that NAFEXP is initially faster than BINEXP on average, as is SB3EXP for the smallest discriminant bit length, but as the size of the discriminant grows, the computational advantage diminishes. We conjecture that, especially in the SB3EXP case, this may be due to the relative costs of WDIV and WCUBE with respect to WMULT and WDUPL; further investigation is required to determine this and reduce any observed discrepancies. Another factor may be that the binary expansion does not require pre-computation or storage, whereas the NAF and SB3EXP must be pre-computed. Using a "right-to-left" variant of NAF may alleviate this problem.

Table 2. Average CPU times (in seconds) per key agreement per partner with Lehmer's partial GCD algorithm for 1000 trials

$\log_2 B$	$\log_2 \Delta$	BINEXP	NAFEXP	% diff	SB3EXP	% diff
224	1341	0.122770	0.109070	11.2	0.119250	2.9
256	1818	0.193320	0.169720	7.8	0.197480	-2.2
384	3586	0.712660	0.674110	5.4	0.823520	-15.6
512	5957	1.932860	2.029010	-5.0	2.593800	-32.4

It is of interest to find out how often Alice and Bob might not compute the same key ideal using the methods described above. For 10 000 trials using a 1341-bit discriminant, no mismatches were found for the ideals computed with any of the three exponentiation methods. The number of mismatches for BINEXP was found to decrease with discriminant size in previous work [15, Table 14.1, p. 369].

We also wished to determine whether the precision bounds established in our analysis herein were tight. Using 200 trials, we determined for each exponentiation method the minimum precision value for which all of the 200 trial key agreements were successful (i.e. established ideals that were within ± 2 baby steps of each other). We found that BINEXP required 15-16 fewer bits of precision than the theoretical lower bound. For NAFEXP we could use 14-17 fewer bits of precision, and for SB3EXP, 17-19 fewer bits of precision were sufficient. The precision results of these trials are listed in [7, Table 5.9]. The 200 precision tests were timed in order to determine if an improved precision analysis could change our efficiency results. We found that the timings with the reduced precision followed the same trends as those of Table 2, and that the performance improvement was negligible. These results are listed in [7, Table 5.10].

8 Conclusions

We investigated methods for exponentiation in the infrastructure of a real quadratic field that were used in a key agreement protocol. We found that using the non-adjacent form exponentiation method for w-near (f, p) representations improved the computing time for most discriminant sizes tested and, according to our analysis, typically did not increase the precision requirements compared to the corresponding binary exponentiation method. However, it remains an open problem to determine why NAFEXP slows in comparison to BINEXP as the discriminant size increases. Our preliminary investigation of this behavior showed that the time devoted to computing the non-adjacent form of the exponents did not fully account for why NAFEXP slowed down in comparison to BINEXP. Similar further work is required for SB3EXP. It remains to be determined whether this observed slowing is a result of the implementation of the methods or if it is a property of the methods themselves.

There are exponentiation methods that improve on NAFEXP. One could employ a sliding window NAF method [10, Algorithm 3.38, p. 101] which uses pre-computations to decrease the number of operations in the actual algorithm. An adaptation of a double-base method such as the ternary/binary method described in [5] would also be of interest as this could limit the use of the cubing steps compared to SB3EXP while still reducing the overall number of operations compared to BINEXP. To use these methods, an investigation of the precision requirements would be needed.

Further optimization of our algorithms and their implementations is possible. For example, the NUCOMP and WNEAR methods could be combined in such a way so that NUCOMP passes certain values to WNEAR, thereby reducing computation effort. Our precision analysis assumed a worst case scenario, so the

required precision is not tight in practice. It is also worthwhile to investigate whether $w = (\log \Delta)/4$ is the best choice of w for SB3EXP; if this choice of w is not optimal, then the performance of our ternary algorithm could be improved.

Recent results from hyperelliptic curve cryptography, particularly those from [12], employ scalar multiplication methods in which a large number of giant steps is replaced by a series of (much faster) baby steps. These could potentially be adapted to the real quadratic field infrastructure setting.

Section 7 established that using NAFEXP improved the efficiency of infrastructure based key agreement compared to using BINEXP, which is not the case, for example, in the original Diffie-Hellman protocol [6]. Improvements of this kind are important for practical use. It is essential to have a varied set of cryptographic protocols for key agreement due to the constant pressure of technology and advancement of cryptographic attacks. The methods developed herein could also be used to develop a signature scheme similar to that of Guillou and Quisquater [9], which would expand the breadth of cryptographic protocols available for use.

References

1. Barker, E., Barker, W., Polk, W., Smid, M.: Recommendation for key management - part 1: General (revised). NIST Special Publication 800-57, National Institute of Standards and Technology (NIST) (March 2007), http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf
2. Biasse, J.-F., Jacobson Jr., M.J., Silvester, A.K.: Security Estimates for Quadratic Field Based Cryptosystems. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 2010. LNCS, vol. 6168, pp. 233–247. Springer, Heidelberg (2010), http://dl.acm.org/citation.cfm?id=1926211.1926229
3. Buchmann, J., Williams, H.C.: A key-exchange system based on imaginary quadratic fields. Journal of Cryptology 1, 107–118 (1988)
4. Buchmann, J., Williams, H.C.: A Key Exchange System Based on Real Quadratic Fields. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 335–343. Springer, Heidelberg (1990), http://dl.acm.org/citation.cfm?id=646754.705067
5. Ciet, M., Joye, M., Lauter, K., Montgomery, P.: Trading inversions for multiplications in elliptic curve cryptography. Designs, Codes and Cryptography 39, 189–206 (2006)
6. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)
7. Dixon, V.: Fast Exponentiation in the Infrastructure of a Real Quadratic Field. Master's thesis, University of Calgary, Calgary, Alberta (2011)
8. Free Software Foundation: The GNU Multiple Precision Arithmetic Library (2011), http://gmplib.org
9. Guillou, L.C., Quisquater, J.-J.: A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 123–128. Springer, Heidelberg (1988)

10. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography, pp. 98–99. Springer Science and Buisness Media, LLC (2004)
11. Imbert, L., Jacobson Jr., M.J., Schmidt, A.: Fast ideal cubing in imaginary quadratic number and function fields. Advances in Mathematics of Communications 4(2), 237–260 (2010)
12. Jacobson Jr., M.J., Scheidler, R., Stein, A.: Cryptographic aspects of real hyperelliptic curves. Tatra Mountains Mathematical Publications 45, 1–35 (2010)
13. Jacobson Jr., M.J., Scheidler, R., Williams, H.C.: The efficiency and security of a real quadratic field based key exchange protocol. In: Public Key Cryptography and Computational Number Theory (Warsaw 2000), pp. 89–112. Walter de Gruyter, Berlin (2001)
14. Jacobson Jr., M.J., Scheidler, R., Williams, H.C.: An improved real quadratic field based key exchange procedure. Journal of Cryptology 19, 211–239 (2006)
15. Jacobson Jr., M.J., Williams, H.C.: Solving the Pell Equation. CMS Books in Mathematics. Springer (2009) iSBN 978-0-387-84922-5
16. Jebelean, T.: A double-digit Lehmer-Euclid algorithm for finding the GCD of long integers. Journal of Symbolic Computation 19, 145–157 (1995)
17. Lehmer, D.H.: Euclid's algorithm for large numbers. The American Mathematical Monthly 45(4), 227–233 (1938)
18. Shanks, D.: The infrastructure of real quadratic fields and its applications. In: Proc. 1972 Number Theory Conf., Boulder, Colorado, pp. 217–224 (1972)
19. Shanks, D.: On Gauss and composition I, II. In: Proceedings NATO ASI on Number Theory and Applications, pp. 163–204. Kluwer, Dordrecht (1989)
20. Silvester, A.: Doctoral Dissertation, University of Calgary (in progress, 2012)

A Appendix

For reference, pseudocode for our new algorithms is listed below. WDUPL, WCUBE, and WDIV all require the sub-algorithms REMOVE (for adjusting the relative generator approximation obtained after the operation) and WNEAR (for adjusting the output to satisfy the definition of a w-near (f,p) representation). The specifications of these algorithms are listed for convenience, together with references to their descriptions. Complete descriptions of the remaining algorithms can be found in [7].

Algorithm A.1. REMOVE [15, Algorithm A.1, p. 448]

Input: $(\mathfrak{b},e,h),T,C,s,p$, where $((\mu)\mathfrak{b},e,h)$ is an (f,p) representation of some ideal $\mathfrak{a}$ with $\mu = |(A+B\sqrt{D})/C| \geq 1$ $(A,B,C \in \mathbb{Z}$ and $C \neq 0)$, $T = 2^sA + B\lfloor 2^s\sqrt{D}\rfloor$, and $s \in \mathbb{Z}^{\geq 0}$ with $2^s|C| > 2^{p+4}|B|$.

Output: An $(f+9/8,p)$ representation $(\mathfrak{b},e',h')$ of $\mathfrak{a}$.

Algorithm A.2. WNEAR [15, Algorithm 11.2, pp. 454-456]

Input: $(\mathfrak{b},d,k),w,p$, where $(\mathfrak{b},d,k)$ is an (f,p) representation of some $\mathcal{O}$-ideal $\mathfrak{a}$. Here, $\mathfrak{b} = \left[Q/r,(P+\sqrt{D})/r\right]$, where $P+\sqrt{D} \geq Q$, $0 \leq \lfloor\sqrt{D}\rfloor - P \leq Q$.

Output: A w-near $(f+9/8,p)$ representation $(\mathfrak{c},g,h)$ of $\mathfrak{a}$.

Algorithm A.3. WDUPL

Input: $(\mathfrak{b}', d', k'), w, p$, where $(\mathfrak{b}', d, k)$ is a w-near (f', p) representation of some ideal $\mathfrak{a}$. Here $\mathfrak{b}' = [Q'/r, (P' + \sqrt{D})/r]$.

Output: A reduced w-near (f, p) representation $(\mathfrak{c}, d, k)$ of $\mathfrak{a}^2$ where $f = 2f' + 2^{-p}(f')^2 + 13/4$.

/* *Finding a reduced* $\mathfrak{b}$ *and* $\mu = |(A + B\sqrt{D})/C| \in \mathbb{K}$ *satisfying* $(\mu)\mathfrak{b} = (\mathfrak{b}')^2$. */

Compute $S = Z(Q'/r) + Y(2P'/r)$ where $S = \gcd(Q'/r, 2P'/r)$ for $S, Y \in \mathbb{Z}$.
Set $N = Q'/(Sr)$, $L = Q'/S$, $K = YR'$ (mod L).
Set $R_{-1} = L$, $R_0 = K$, $C_{-1} = 0$, $C_0 = -1$ and $i = 0$.
if $R_{-2} < \lfloor \sqrt{2rD^{1/2}} \rfloor$ **then**
  Put $Q_{i+1} = (Q')^2/(rS^2)$ and $P_{i+1} \equiv P' + YR'Q'/(rS)$ (mod Q_{i+1}).
else
  while $R_i > \sqrt{2r}|D|^{1/4}$ **do**
    $i \leftarrow i + 1$
    $q = \lfloor R_{i-2}/R_{i-1} \rfloor$
    $R_i = R_{i-2} - q_i R_{i-1}$
    $C_i = C_{i-2} - q_i C_{i-1}$
  end while
  $M_2 = (R_i 2P' + rSR'C_i)/L$
  $Q_{i+1} = (-1)^{i-1}(R_i^2/r - C_i M_2)$
  $P_{i+1} = (NR_i + Q_{i+1}C_{i-1})/C_i - P'$
end if
/* *Final reduction steps* */
$j = 1$
$Q'_{i+1} = |Q_{i+1}|$
$k_{i+1} = \lfloor (\sqrt{D} - P_{i+1})/Q'_{i+1} \rfloor$
$P'_{i+1} = k_{i+1}Q'_{i+1} + P_{i+1}$
$\sigma = \mathrm{sign}(Q_{i+1})$
$B_{i-1} = \sigma|C_{i-1}|$
$B_{i-2} = |C_{i-2}|$.
if $P'_{i+1} + \lfloor \sqrt{D} \rfloor < Q'_{i+1}$ **then**
  $j = 2$
  $q_{i+1} = \lfloor (P_{i+1} + \lfloor \sqrt{D} \rfloor)/Q'_{i+1} \rfloor$
  $P_{i+2} = q_{i+1}Q'_{i+1} - p_{i+1}$, $Q_{i+2} = (D - P_{i+2}^2)/Q'_{i+1}$
  $Q'_{i+2} = |Q_{i+2}|$
  $k_{i+2} = \lfloor (\sqrt{D} - P_{i+2})/Q'_{i+2} \rfloor$
  $P'_{i+2} = k_{i+2}Q'_{i+2} + P_{i+2}$
  $B_{i+1} = q_{i+1}B_i + B_{i+1}$.
  if $P'_{i+2} + \lfloor \sqrt{D} \rfloor < Q'_{i+2}$ **then**
    $j = 3$
    $q_{i+2} = \lfloor (P_{i+2} + \lfloor \sqrt{D} \rfloor)/Q_{i+2} \rfloor$
    $P_{i+3} = q_{i+2}Q_{i+2} - p_{i+2}$

36: $Q_{i+3} = (D - P_{i+3}^2)/Q_{i+2}$
37: $Q'_{i+3} = |Q_{i+3}|$
38: $k_{i+3} = \lfloor(\sqrt{D} - P_{i+3})/Q'_{i+3}\rfloor$
39: $P'_{i+3} = k_{i+3}Q'_{i+3} + P_{i+3}$
40: $B_{i+2} = q_{i+2}B_{i+1} + B_{i+2}$.
41: **end if**
42: **end if**
43: Put $\mathfrak{b} = [Q'_{i+j}/r, (P'_{i+1} + \sqrt{D})/r]$, $A = S(Q_{i+j}B_{i+j-2} + P_{i+j}B_{i+j-1})$, $B = -SB_{i+j-1}$, and $C = Q_{i+j}$.

/ Using Theorem 1 with equal inputs */*
44: **if** $(d')^2 \leq 2^{2p+1}$ **then**
45: Put $e = \lfloor(d')^2/2^p\rfloor$, $h = 2k'$.
46: **else**
47: Put $e = \lfloor(d')^2/2^{p+1}\rfloor$, $h = 2k' + 1$.
48: **end if**

/ Bounding μ and calling REMOVE. */*
49: Find $s \geq 0$ such that $2^sQ > 2^{p+4}B$ and put $T = 2^sA + B\lfloor 2^s\sqrt{D}\rfloor$.
50: $(\mathfrak{b}, e', h') = \text{REMOVE}((\mathfrak{b}, e, d), T, C, s, p)$.

/ Calling WNEAR */*
51: $(\mathfrak{c}, d, k) = \text{WNEAR}((\mathfrak{b}, e', h'), w, p)$

Algorithm A.4. WCUBE

Input: $(\mathfrak{b}', d', k')$, w, p where $(\mathfrak{b}', d', k')$ is a reduced w-near (f', p) representation of an invertible $\mathcal{O}$-ideal $\mathfrak{a}'$. Here $\mathfrak{b}'' = \left[Q'/r, (P' + \sqrt{D})/r\right]$.

Output: A w-near $(3f' + 3(f')^2 2^{-p} + (f')^3 2^{-2p} + 13/4, p)$ representation $(\mathfrak{c}, d, k)$ of $(\mathfrak{a}')^3$.

/ Finding a reduced* $\mathfrak{b}$ *and* $\mu = (A + B\sqrt{D})/C \in \mathbb{K}$ *for which* $(\mu)\mathfrak{b} = (\mathfrak{b}')^2$. *Lines 1-22 are [11, Algorithm 4, p. 16]. */*

1: Find $S', v_1 \in \mathbb{Z}$ such that $S' = \gcd(Q'/r, 2P'/r)$ and $S' = u_1(Q'/r) + v_1(2P'/r)$.
2: **if** $S' = 1$ **then**
3: Set $S = 1$, $N = Q'/r$, $L = NQ'$, and $K = R'v_1(2 - v_1(v_1(Q'/r)(R'/r) + (2P')/r)) \pmod{L}$.
4: **else**
5: Compute $S = u_2(S'Q'/r) + v_2((3(P')^2 + D)/r^2)$, $N = Q'/(rS)$, $L = NQ'$, and $K = R'(u_2v_1(Q'/r) + v_2(2P'/r)) \pmod{L}$.
6: **end if**
7: **if** $L < \sqrt{Q'/r^2}|D|^{1/4}$ **then**
8: Set $Q = NL$ and $P = P' + NK$.
9: **else**
10: Set $R_{-1} = L, R_0 = K, C_{-1} = 0, C_0 = -1, i = 0$.
11: **while** $R_i > \sqrt{Q'/r^2}|D|^{1/4}$ **do**
12: $i \leftarrow i + 1$
13: $q_i = \lfloor R_{i-2}/R_{i-1} \rfloor$
14: $R_i = R_{i-2} - q_i R_{i-1}$
15: $C_i = C_{i-2} - q_i C_{i-1}$
16: **end while**
17: $P'' = P' + NK \pmod{L}$
18: $M_1 = (NR_i + (P'' - P')C_i)/L$
19: $M_2 = (R_i(P' + P'') + rR'SC_i)/L$
20: $Q = (-1)^{i-1}(R_iM_1 - C_iM_2)$
21: $P = (NR_i + QC_{i-1})/C_i - P'$
22: **end if**

/ Final reduction steps (using the same method as NUCOMP). */*

23: $j = 1$
24: $Q'_{i+1} = Q_i = |Q|$
25: $k = \lfloor (\sqrt{D} - P_{i+1})/Q'_{i+1} \rfloor$
26: $P'_{i+1} = k_{i+1}Q'_{i+1} + P_{i+1}$
27: $\sigma = \text{sign}(Q)$
28: $B_{i-1} = \sigma|C_{i-1}|$
29: $B_i = |C_i|$
30: **if** $P'_{i+1} + \lfloor \sqrt{D} \rfloor < Q'_{i+1}$ **then**
31: $j = 2$

32: $q_{i+1} = \lfloor (P_{i+1} + \lfloor \sqrt{D} \rfloor)/Q'_{i+1} \rfloor$
33: $P_{i+2} = q_{i+1}Q'_{i+1} - p_{i+1}$, $Q_{i+2} = (D - P^2_{i+2})/Q'_{i+1}$
34: $Q'_{i+2} = |Q_{i+2}|$
35: $k_{i+2} = \lfloor (\sqrt{D} - P_{i+2})/Q'_{i+2} \rfloor$
36: $P'_{i+2} = k_{i+2}Q'_{i+2} + P_{i+2}$
37: $B_{i+1} = q_{i+1}B_i + B_{i+1}$.
38: **if** $P'_{i+2} + \lfloor \sqrt{D} \rfloor < Q'_{i+2}$ **then**
39: $j = 3$
40: $q_{i+2} = \lfloor (P_{i+2} + \lfloor \sqrt{D} \rfloor)/Q_{i+2} \rfloor$
41: $P_{i+3} = q_{i+2}Q_{i+2} - p_{i+2}$
42: $Q_{i+3} = (D - P^2_{i+3})/Q_{i+2}$
43: $Q'_{i+3} = |Q_{i+3}|$
44: $k_{i+3} = \lfloor (\sqrt{D} - P_{i+3})/Q'_{i+3} \rfloor$
45: $P'_{i+3} = k_{i+3}Q'_{i+3} + P_{i+3}$
46: $B_{i+2} = q_{i+2}B_{i+1} + B_{i+2}$.
47: **end if**
48: **end if**
49: Put $\mathfrak{b} = [Q'_{i+j}/r, (P'_{i+1} + \sqrt{D})/r]$, $A = S(Q_{i+j}B_{i+j-2} + P_{i+j}B_{i+j-1})$, $B = -SB_{i+j-1}$, and $C = Q_{i+j}$.

/ Using Theorem 2 */*
50: **if** $(d')^3 \leq 2^{3p+1}$ **then**
51: Put $e = \lfloor (d')^3/2^{2p} \rfloor$ and $h = 3k'$.
52: **else if** $2^{3p+1} < (d')^3 \leq 2^{3p+2}$ **then**
53: Put $e = \lfloor (d')^3/2^{2p+1} \rfloor$ and $h = 3k' + 1$.
54: **else**
55: Put $e = \lfloor (d')^3/2^{2p+2} \rfloor$ and $h = 3k' + 2$.
56: **end if**

/ Bounding μ and calling REMOVE */*
57: Find $s \geq 0$ such that $2^sQ > 2^{p+4}B$ and put $T = 2^sA + B\lfloor 2^s\sqrt{D} \rfloor$.
58: $(\mathfrak{b}, e', h') = \text{REMOVE}((\mathfrak{b}, e, h), T, C, s, p)$.

/ Re-establishing the w-near property */*
59: $(\mathfrak{c}, d, k) = \text{WNEAR}((\mathfrak{b}, e', h'), w, p)$.

Algorithm A.5. WDIV

Input: $(\mathfrak{b}', d', k')$, $(\mathfrak{b}'', d'', k'')$, p where $(\mathfrak{b}', d', k')$ is a w-near (f', p) representation of an invertible $\mathcal{O}$-ideal $\mathfrak{a}'$ and $(\mathfrak{b}'', d'', k'')$ is a w-near (f'', p) representation of an invertible $\mathcal{O}$-ideal $\mathfrak{a}''$ dividing $\mathfrak{a}'$. Here,

$$\mathfrak{b}' = [Q'/r, (P' + \sqrt{D})/r] \text{ and } \mathfrak{b}'' = [Q''/r, (P'' + \sqrt{D})/r].$$

Output: $(\mathfrak{c}, d, k)$, a reduced $(f^{**} + 13/4, p)$ representation of $\mathfrak{a}'(\mathfrak{a}'')^{-1}$ where $f^{**} = f' + f''/(1 - f''2^{-p}) + f'f''/(2^p - f'')$.

/ Computing a reduced ideal* $\mathfrak{b}$ *and* $\mu = (A + B\sqrt{D})/C \in \mathbb{K}$ *where* $(\mu)\mathfrak{b} = \mathfrak{b}'\overline{\mathfrak{b}''}$. **/*

1: Let $\mathfrak{b}^* = [Q''/r, (qQ'' - P'' + \sqrt{D})/r]$ where $q = \lfloor (P'' + \sqrt{D})/Q'' \rfloor$.
2: **if** $Q' \geq Q''$ **then**
3: Compute $(\mathfrak{b}, A, B, C) = \text{NUCOMP}(\mathfrak{b}', \mathfrak{b}^*)$ where $\mathfrak{b} = [Q/r, (P+\sqrt{D})/r]$.
4: **else**
5: Compute $(\mathfrak{b}, A, B, C) = \text{NUCOMP}(\mathfrak{b}^*, \mathfrak{b}')$ where $\mathfrak{b} = [Q/r, (P+\sqrt{D})/r]$.
6: **end if**

/ Computing* e, h *for which* $(\mathfrak{b}''\overline{\mathfrak{b}''}, e, h)$ *is an* $(1 + f^{**}, p)$ *representation of* $\mathfrak{a}(\mathfrak{a}'')^{-1}$ *using Theorem 3. */*

7: Find κ such that $2^\kappa < N(\mathfrak{b}'') = Q''/r \leq 2^{\kappa+1}$.
8: **if** $d''2^{\kappa-1} < d'N(\mathfrak{b}'') \leq d''2^\kappa$ **then**
9: $e = \lceil 2^{p-\kappa+1} d'N(\mathfrak{b}'')/d'' \rceil$
10: $h = k' - k'' + \kappa - 1$
11: **else if** $d''2^\kappa < d'N(\mathfrak{b}'') \leq d''2^{\kappa+1}$ **then**
12: $e = \lceil 2^{p-\kappa} d'N(\mathfrak{b}'')/d'' \rceil$
13: $h = k' - k'' + \kappa$
14: **else**
15: $e = \lceil 2^{p-\kappa-1} d'N(\mathfrak{b}'')/d'' \rceil$
16: $h = k' - k'' + \kappa + 1$.
17: **end if**

/ Computing* e', h' *for which* $(\mathfrak{b}, e', h')$ *is an* $(17/8 + f^{**}, p)$ *representation of* $\mathfrak{a}'(\mathfrak{a}'')^{-1}$. **/*

18: Find $s \geq 0$ such that $2^s Q > 2^{p+4} B$.
19: Put $T = 2^s A + B\lfloor 2^s \sqrt{D} \rfloor$.
20: $(\mathfrak{b}, e', h') = \text{REMOVE}((\mathfrak{b}, e, h), T, C, s, p)$

/ Computing a reduced* w*-near* $(13/4 + f^{**}, p)$ *representation of* $\mathfrak{a}'(\mathfrak{a}'')^{-1}$. **/*

21: $(\mathfrak{c}, d, k) = \text{WNEAR}((\mathfrak{b}, e', h'), w, p)$

Algorithm A.6. BINEXP [15, Algorithm 11.4, pp. 276-277]

Input: $(\mathfrak{b}_0, d_0, k_0), n, w, p$ where $(\mathfrak{b}_0, d_0, k_0)$ is a w-near (f_0, p) representation of an invertible $\mathcal{O}$-ideal $\mathfrak{a}$ and $n \in \mathbb{N}$.

Output: A w-near (f, p) representation $(\mathfrak{b}, d, k)$ of $\mathfrak{a}^n$ for some $f \in [1, 2^p)$.

Compute $(n_0, \ldots, n_{\ell_B}) = \text{BIN}(n)$.
Set $(\mathfrak{b}, d, k) = (\mathfrak{b}_0, d_0, k_0)$.
for $i = 1 \to \ell_B$ **do**
  $(\mathfrak{b}, d, k) \leftarrow$WDUPL$((\mathfrak{b}, d, k), w, p)$
  if $n_i = 1$ **then**
    $(\mathfrak{b}, d, k) \leftarrow$WMULT$((\mathfrak{b}, d, k), (\mathfrak{b}_0, d_0, k_0), w, p)$
  end if
end for

Algorithm A.7. NAFEXP

Input: $(\mathfrak{b}_0, d_0, k_0), n, w, p$ where $(\mathfrak{b}_0, d_0, k_0)$ is a w-near (f_0, p) representation of an invertible $\mathcal{O}$-ideal $\mathfrak{a}$ and $n \in \mathbb{N}$.

Output: A w-near (f, p) representation $(\mathfrak{b}, d, k)$ of $\mathfrak{a}^n$ for some $f \in [1, 2^p)$.

Compute $(n_0, \ldots, n_{\ell_N}) = \text{NAF}(n)$.
Set $(\mathfrak{b}, d, k) = (\mathfrak{b}_0, d_0, k_0)$.
for $i = 1 \to \ell_N$ **do**
  $(\mathfrak{b}, d, k) \leftarrow$WDUPL$((\mathfrak{b}, d, k), w, p)$
  if $n_i = 1$ **then**
    $(\mathfrak{b}, d, k) \leftarrow$WMULT$((\mathfrak{b}, d, k), (\mathfrak{b}_0, d_0, k_0), w, p)$
  else if $n_i = -1$ **then**
    $(\mathfrak{b}, d, k) \leftarrow$WDIV$((\mathfrak{b}, d, k), (\mathfrak{b}_0, d_0, k_0), w, p)$
  end if
end for

Algorithm A.8. SB3EXP

Input: $(\mathfrak{b}_0, d_0, k_0), n, w, p$ where $(\mathfrak{b}_0, d_0, k_0)$ is a w-near (f_0, p) representation of an invertible $\mathcal{O}$-ideal $\mathfrak{a}$ and $n \in \mathbb{N}$.

Output: A w-near (f, p) representation $(\mathfrak{b}, d, k)$ of $\mathfrak{a}^n$ for some $f \in [1, 2^p)$.

Compute $(n_0, \ldots, n_{\ell_S}) = \text{SB3}(n)$.
Set $(\mathfrak{b}, d, k) = (\mathfrak{b}_0, d_0, k_0)$.
for $i = 1 \to \ell_S$ **do**
  $(\mathfrak{b}, d, k) \leftarrow$WCUBE$((\mathfrak{b}, d, k), w, p)$
  if $n_i = 1$ **then**
    $(\mathfrak{b}, d, k) \leftarrow$WMULT$((\mathfrak{b}, d, k), (\mathfrak{b}_0, d_0, k_0), w, p)$
  else if $n_i = -1$ **then**
    $(\mathfrak{b}, d, k) \leftarrow$WDIV$((\mathfrak{b}, d, k), (\mathfrak{b}_0, d_0, k_0), w, p)$
  end if
end for

UOWHFs from OWFs: Trading Regularity for Efficiency

Kfir Barhum and Ueli Maurer

Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
{barhumk,maurer}@inf.ethz.ch

Abstract. A universal one-way hash function (UOWHF) is a shrinking function for which finding a second preimage is infeasible. A UOWHF, a fundamental cryptographic primitive from which digital signature can be obtained, can be constructed from any one-way function (OWF). The best known construction from any OWF $f : \{0,1\}^n \to \{0,1\}^n$, due to Haitner et. al. [2], has output length $\tilde{O}(n^7)$ and $\tilde{O}(n^5)$ for the uniform and non-uniform models, respectively. On the other hand, if the OWF is known to be injective, i.e., maximally regular, the Naor-Yung construction is simple and practical with output length linear in that of the OWF, and making only one query to the underlying OWF.

In this paper, we establish a trade-off between the efficiency of the construction and the assumption about the regularity of the OWF f. Our first result is a construction comparably efficient to the Naor-Yung construction but applicable to any close-to-regular function. A second result shows that if $|f^{-1}f(x)|$ is concentrated on an interval of size $2^{s(n)}$, the construction obtained has output length $\tilde{O}(n \cdot s(n)^6)$ and $\tilde{O}(n \cdot s(n)^4)$ for the uniform and non-uniform models, respectively.

Keywords: Complexity-Based Cryptography, One-Way Functions, Universal One-Way Hash Functions, Computational Entropy.

1 Introduction

1.1 Constructions of Cryptographic Primitives

A main task in cryptographic research is to construct a (strong) cryptographic primitive P from a (weaker) cryptographic primitive Q, for example to construct a pseudo-random generator from a one-way function (OWF). This paper is concerned with constructing a universal one-way hash function (UOWHF), a fundamental cryptographic primitive, from a OWF.

The term "construct" means that one gives an efficient reduction of the problem of breaking the underlying primitive Q to the problem of breaking the constructed primitive P. For two primitives P and Q, the most basic question is whether P can be constructed *in principle* from Q, meaning that the construction and the reduction must be efficient (i.e., polynomial-time) and that the reduction translates a non-negligible probability of breaking P into a non-negligible probability of breaking Q.

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 234–253, 2012.

The principle possibility of constructing a UOWHF from a OWF was proved by Rompel [7], using a highly inefficient construction and reduction. When trying to improve the construction, one can choose two orthogonal routes. Either one improves the construction for a general OWF, or one makes specific assumptions about the OWF allowing for special-purpose constructions that do not necessarily work in general, and can hence be more efficient. Of course, a key issue is how restrictive or how plausible the assumption one has to make is.

The best known *general* construction of a universal one-way hash function from any one-way function $f : \{0,1\}^n \to \{0,1\}^m$, due to Haitner et. al. [2], has output length $\tilde{O}(n^7)$ and $\tilde{O}(n^5)$ for the uniform and non-uniform cases, respectively. The best known special-purpose construction is due to Naor and Yung [6] and makes a single call to f (per argument to the constructed UOWHF), and the output length is linear in n, but the assumption one needs to make is that f is *injective*, which is a very strong assumption.

In this paper we investigate the middle grounds between completely general constructions and those requiring such a very specific assumption. Concretely, we investigate the trade-off between the regularity assumption for f and the efficiency of the construction. The regularity is characterized by how concentrated the *preimage size spectrum*, the random variable $|f^{-1}(f(X))|$ corresponding to the preimage size of the function value $f(X)$ for a uniformly random argument X, is. For injective functions, the preimage size spectrum is constant 1. Prior to our work, we do not know of any specific construction for a function which is anywhere between regular and arbitrary.

In this work we relate the assumptions made about the spectrum of f to the efficiency of the overall construction. Qualitatively speaking, the more is assumed about the regularity of f, the more efficient is the resulting construction.

1.2 Contributions of This Paper

A first result on the way to fully utilizing an assumption about the regularity of a function is an almost optimal construction of a universal one-way hash function from a regular (or almost regular) one-way function. Recall that a function is 2^r-regular if for every image there are 2^r preimages.

Following previous work, for simplicity of presentation, we assume that for a one-way function f the input length n is the security parameter. For this case, we get a construction with output length and key length $O(n \cdot \alpha(n) \cdot \log(n))$, where the construction makes $O(\alpha(n) \cdot \log(n))$ invocations to f for any super-constant function $\alpha(n)$. This improves on [8] by a factor of $\log(n)$ (see Section 1.3 for comparison with previous work).

We introduce a natural relaxation of the notion of regularity:

Definition 1 (roughly-regular function). *A function $f : \{0,1\}^n \to \{0,1\}^m$ is called (r,s)-*roughly-regular*, if for every x in $\{0,1\}^n$ it holds that $|f^{-1}(f(x))|$ lies in the interval $[r, rs]$. A family of functions $f = \{f_n\}_{n>0}$ is called (r,s)-*roughly-regular*, where $(r,s) = (r(n), s(n))$, if for every n it holds that f_n is $(r(n), s(n))$-roughly regular. Whenever $s(n) = n^c$ for some constant c the family is called r-*polynomially-roughly-regular*.*

We call r and s the *regularity* and the *roughness* parameters of f, respectively. Indeed, whenever the roughness parameter is trivial, that is, $s(n) = 1$ for all n, this definition coincides with the standard definition of an r-regular function. This definition, we argue, is both intuitive and quantifies the irregularity of a function.

For the case where f is a 2^r-polynomially-roughly-regular OWF, we observe that the construction for the 2^r-regular case with minor changes works (we omit the details in this extended abstract). For a pseudo-random generator based on a regular OWF an analog relaxation was already observed by [1].

Finally, in Section 4 we utilize the ideas developed in Section 3 and improve on [2] with the most general version (Theorem 3). We establish a trade-off between the regularity assumption made about the underlying one-way function and the overall efficiency of the construction. When f is a $(2^{r(n)}, 2^{s(n)})$-roughly regular one-way function, we show a construction with output length and key length of $\tilde{O}(n \cdot s^4)$ for the non-uniform model and of $\tilde{O}(n \cdot s^6)$ for the uniform model. Indeed, our construction ties up both ends of the existing constructions: When s is constant, we get an almost linear construction, and when $s = O(n)$ our construction matches that of [2].

The analysis of the construction presented in Section 3 improves by a factor of $O(\log^2(n))$ on the construction presented in Section 4 when instantiated with a 2^r-regular function.

1.3 Related Work

Inaccessible Entropy. Our work uses the framework of [2] for constructing UOWHFs from OWFs using the notion of inaccessible entropy. Inaccessible entropy was first introduced in [5] and along with work done in [3] and [4], it completes the construction of the fundamental cryptographic primitives: universal one-way hash functions, pseudo-random generators and commitment schemes using this notion.

A Regularity-Efficiency Trade-Off for the Construction of a UOWHF. In [8] it was first shown how to construct a UOWHF for the almost-regular case. Our construction achieves the same query complexity to the underlying one-way function ($O(\alpha(n) \cdot \log(n))$ calls), but is superior in two aspects: It makes its queries to the underlying one-way function in a non-adaptive manner, and our resulting primitive has an output (and seed) length of $n \cdot log(n) \cdot \alpha(n)$ whereas the construction from [8] has an additional $\log(n)$ factor.

While for the almost-regular case the improvement is not dramatic, we believe that our analysis, which extends the approach suggested in [2], sheds more light on what is achieved at each step. The way the almost-regularity property of the underlying one-way function is utilized later allows to generalize it to any level of regularity. This is in contrast to the construction in [8] which is more ad-hoc.

2 Preliminaries

2.1 Notations and Basics

Throughout the paper we use capital letters to denote random variables and small letters for specific values they assume. We denote by $\mathcal{N}$ the set of natural numbers. For an integer n we denote by $[n]$ the set $\{1,\ldots,n\}$. For two bit-strings x and y we denote their concatenation by $x\|y$. For a random variable X we denote by $\mathbf{E}[X]$ and $\mathbf{V}[X]$ its expectation and variance, accordingly. For an event A we denote its indicator random variable (which assumes the value 1 whenever A happens and 0 otherwise) by $\mathbb{1}_A$, and its complement event by $\overline{A}$. We implicitly make use of the fact that $\mathbf{E}[\mathbb{1}_A] = \Pr[A]$. The support of a random variable X is defined as $\mathrm{Supp}(X) = \{x : \Pr[X = x] > 0\}$. For a function $f : X \to Y$, we define the preimage spectrum function $\pi_f : X \to \mathcal{N}$, where $\pi_f(x) = |f^{-1}(f(x))|$.

For understood $Y_1 \times \cdots \times Y_n$ we denote by $\phi_i : Y_1 \times \cdots \times Y_n \to Y_i$ the projection onto the i'th component. We extend this to a set $S \subseteq Y_1 \times \cdots \times Y_n$ with $\phi_i(S) \stackrel{\text{def}}{=} \{\phi_i(s) : s \in S\}$. A non-decreasing function $f : \mathcal{N} \to \mathcal{N}$ is called super-constant if for all $c \in \mathcal{N}$ there exists an $n \in \mathcal{N}$, such that $f(n) > c$. All log functions are to the base 2.

We cite the Hoeffding bound and bring the definition of a t-wise independent hash family in Appendix A.

2.2 OWF and UOWHF

Definition 2 (OWF). *A family of functions $\{f : \{0,1\}^{n(\rho)} \to \{0,1\}^{m(\rho)}\}_{\rho\in\mathcal{N}}$, where ρ is a security parameter, is a* one-way function *if:*

1. *There exists an efficient algorithm that, given x, evaluates $f(x)$.*
2. *For any efficient randomized algorithm A:*

$$\Pr_{x\xleftarrow{r}\{0,1\}^{n(\rho)}}\left[A(1^\rho, f(x)) \in f^{-1}(f(x))\right] \leq \mathsf{negl}(\rho) \ .$$

Definition 3 (UOWHF). *A family of keyed functions $\{\{f_k : \{0,1\}^{n(\rho)} \to \{0,1\}^{m(\rho)}\}_{k\in K(\rho)}\}_{\rho\in\mathcal{N}}$, where ρ is a security parameter, is a* universal one-way hash function *if:*

1. *There exists an efficient algorithm that, given x and k, evaluates $f_k(x)$.*
2. *$m(\rho) < n(\rho)$.*
3. *For any pair of efficient randomized algorithms (A_1, A_2):*

$$\Pr_{k\xleftarrow{r}K(\rho),(x,\sigma)\xleftarrow{r}A_1(1^\rho)}\left[A_2(x,k,\sigma) \in {f_k}^{-1}(f_k(x)) \setminus \{x\}\right] \leq \mathsf{negl}(\rho) \ .$$

As noted, we focus on families of functions where the input domain parameter n equals to the security parameter, i.e., $n(\rho) = \rho$, in which case we parameterize the family by n. Additionally, we slightly abuse notation when referring to a function $f : \{0,1\}^n \to \{0,1\}^{m(n)}$, where formally $f = \{f_n : \{0,1\}^n \to \{0,1\}^{m(n)}\}_{n\in\mathcal{N}}$ is a parametrized family of functions, and often we omit the security parameter when referring to f_n or other parametrized values.

2.3 Entropy Measures

For a random variable X and $x \in \mathrm{Supp}(X)$ the point-wise entropy of X is $H_X(x) \stackrel{\text{def}}{=} -\log(\Pr[X = x])$. The Shannon entropy $H(X)$ and min-entropy $H_\infty(X)$ of X are defined as:

$$H(X) \stackrel{\text{def}}{=} \mathbf{E}\left[H_X(X)\right] \ , \qquad H_\infty(X) \stackrel{\text{def}}{=} -\log\left(\max_{x \in \mathrm{Supp}(X)} \Pr[X = x]\right) \ .$$

These measures extend naturally to the case of a joint distribution of two random variables X, Y. Namely, the conditional point-wise entropy for $(x, y) \in \mathrm{Supp}(X, Y)$ is $H_{X|Y}(x, y) \stackrel{\text{def}}{=} -\log(\Pr[X = x|Y = y])$ and the conditional Shannon entropy is

$$H(X|Y) = \mathop{\mathbf{E}}_{(x,y) \stackrel{r}{\leftarrow} (X,Y)} [H_{X|Y}(x, y)] = \mathop{\mathbf{E}}_{y \stackrel{r}{\leftarrow} Y} [H(X|Y = y)] = H(X, Y) - H(Y).$$

The next definition measures the average and absolute guarantees as for the preimage-size of f in terms of entropy bits.

Definition 4 (preimage entropy measures). *For a function $f : \{0,1\}^n \to \{0,1\}^{m(n)}$, define its* real preimage-entropy *as $H_p(f) \stackrel{\text{def}}{=} H(X|f(X))$, where X is uniformly distributed on $\{0,1\}^n$. f has* min-preimage-entropy *at least $k = k(n)$ (and denote this by $H_{p,\min}(f) \geq k$), if there is a negligible function $\epsilon = \epsilon(n)$ such that*

$$\Pr_{x \stackrel{r}{\leftarrow} \{0,1\}^n} [H_{X|f(X)}(x, f(x)) \geq k] \geq 1 - \epsilon.$$

As the argument X in the definition is uniform, we have that for all x it holds that $H_{X|f(X)}(x, f(x)) = \log(\pi_f(x))$.

2.4 Collision Finders and Accessible Entropy

Definition 4 captures the average and absolute preimage set size guarantees for f. Clearly, when f is shrinking it has high preimage-entropy. Recall that our goal is to build a universal one-way hash function, namely, a shrinking function for which there exist many preimages, but at the same time any efficient algorithm, when given an x, cannot compute a different preimage from $f^{-1}(f(x))$.

Definition 5 (f-collision-finder). *Let $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ be a function. An f-collision-finder is a randomized algorithm A such that $A(x) \in f^{-1}(f(x))$ for every $x \in \{0,1\}^n$.*

The requirement that $A(x)$ outputs a preimage of $f(x)$ can be made without loss of generality, as every algorithm A can be changed to one that outputs x whenever $A(x) \notin f^{-1}(f(x))$.

Using the notion of an f-collision-finder, one can define a computational analog of the definitions of real- and min-preimage-entropy of f. The analogous definitions capture the maximal, average, and absolute size of the preimage sets that are accessible to any efficient algorithm.

Definition 6. *A function* $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ *has* accessible max-preimage-entropy *at most* $k = k(n)$ *if there exists a family of sets* $\{S_x\}_{x\in\{0,1\}^n}$ *such that for any efficient randomized* f*-collision-finder* A*, there exists a negligible function* $\epsilon = \epsilon(n)$ *such that for all sufficiently large* n*:*

1. $\Pr_{x\xleftarrow{r}\{0,1\}^n}[A(x) \in S_x] \geq 1 - \epsilon.$
2. $\log(|S_x|) \leq k$ *for all* x*.*

Definition 7. *A function* $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ *has* accessible average max-preimage-entropy *at most* $k = k(n)$ *if it satisfies Definition 6 where instead of (2.) we have:*

2. $\mathbf{E}_{x\xleftarrow{r}\{0,1\}^n}[\log(|S_x|)] \leq k.$

We stress that these two definitions[1] are different from the classical definitions of Shannon entropy. As they capture the inputs accessible only to *efficient* algorithms, both definitions only bound from above the performance of such algorithms. Specifically, for an arbitrary function, we do not know how to compute exactly (as in the standard definition of entropy) these bounds. Nevertheless, as we see next, these bounds are a useful tool (see also [5]). We use the notation $H^{\text{eff}}_{p,\max}(f) \leq k$ and $H^{\text{eff}}_{p,\text{avg}-\max}(f) \leq k$ to denote that the corresponding bound holds.

The next two definitions are used to distinguish between two types of 'entropy gaps':

Definition 8. *A function* $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ *has an* average inaccessible preimage-entropy gap $\Delta = \Delta(n)$*, if there exists some* $k = k(n)$ *such that:*

$$H^{\text{eff}}_{p,\text{avg}-\max}(f) \leq k \leq k + \Delta \leq H_p(f) \ . \tag{1}$$

That is, there is a gap of Δ between its average accessible max-preimage-entropy and its preimage-entropy. At times we will refer to this gap as an average entropy gap or a weak type of gap.

Definition 9. *A function* $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ *has an* absolute inaccessible preimage-entropy gap $\Delta = \Delta(n)$*, if there exists some* $k = k(n)$ *such that:*

$$H^{\text{eff}}_{p,\max}(f) \leq k \leq k + \Delta \leq H_{p,\min}(f) \ . \tag{2}$$

[1] In fact, one may consider a weaker notion of algorithm-dependent accessible max-preimage-entropy and algorithm-dependent accessible average max-preimage-entropy where the sets $\{S_x\}$ may also depend on the algorithm. Such a definition would only require that for every algorithm there exist sets $\{S_{A,x}\}$. This weaker variant of Definitions 6 and 7 is enough for the purpose of constructing a universal one-way hash function and potentially may be easier to satisfy. In this work we do not make use of the weaker definition.

At times we will refer to this gap as an absolute or strong gap.

An important observation is that UOWHFs are just length-decreasing functions with accessible max-preimage-entropy 0, and an appropriate absolute entropy gap. Haitner et. al. observed that it is possible to achieve a noticeable gap of inaccessible entropy as an intermediate step, and then amplify it and transform it into a UOWHF.

2.5 Entropy Measures for t-fold Parallel Repetitions

For a function $f : X \to Y$ we define its t-fold parallel repetition $f^t : X^t \to Y^t$ as $f^t(x_1, \ldots, x_t) = (f(x_1), \ldots, f(x_t))$. It is well-known that using the definition of conditional entropy, properties of $\log(\dot{})$ and noting that choosing a random $x^t \in X^t$ can be done by t independent choices of x,

$$H_p(f^t) = H(X_1, \ldots, X_t | f(X_1), \ldots, f(X_t)) = H(X_1|f(X_1)) + \ldots \\ +H(X_t|f(X_t)) = t \cdot H_p(f) \ . \qquad (3)$$

The corresponding computational bound is given by the following claim and its corollary. Namely, the accessible preimages of the t-fold repetition of f come from the product set of the accessible preimages set of f:

Lemma 1. *Let $f : X \to X$ with accessible max-preimage-entropy at most $k(n)$, with sets S_x (as in Definition 6). Then for $t = \text{poly}(n)$ any efficient f^t-collision-finder A' outputs a collision (except with negligible probability) from the set $S_{x^t} \stackrel{\text{def}}{=} S_{x_1} \times \cdots \times S_{x_t}$.*

Proof. Let A' be an f^t-collision-finder algorithm with probability ϵ to output a collision $x'_1, \ldots, x'_t$ outside of S_{x^t}. Observe that this implies that for a randomly chosen coordinate $i \stackrel{r}{\leftarrow} [t]$ it holds that $\Pr[\phi_I(f^t(X^t)) \notin S_{\phi_I(X^t)}] \geq \epsilon/t$. This calls for the following f-collision-finder A: on input x choose uniformly at random a location i from $[t]$ and uniformly at random inputs $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_t$ from X. Set $x_i = x$ and return $\phi_i(A'(x_1, \ldots, x_t))$. It follows that A outputs a collision for f outside of S_x with probability greater than ϵ/t. The lemma follows. □

Using linearity of expectation, the union bound, Definitions 6 and 7, and the fact that $\log(|S_{x^t}|) = \Sigma_{i=1}^t \log(|S_{x_i}|)$, we get:

Corollary 1.

1. *If $H^{\text{eff}}_{p,\max}(f) \leq k$ then $H^{\text{eff}}_{p,\max}(f^t) \leq t \cdot k$.*
2. *If $H^{\text{eff}}_{p,\text{avg}-\max}(f) \leq k$ then $H^{\text{eff}}_{p,\text{avg}-\max}(f^t) \leq t \cdot k$.*

2.6 An Overview of the Construction of Haitner et. al.

The construction consists of two independent parts. First they show how to get a function with a noticeable gap of average inaccessible entropy from any one-way

function. Specifically, they show that a prefix of a random length of a three-wise independent hashing of the output already has some weak form of an average entropy gap. Namely, on average over the inputs to the new construction, there is a noticeable gap of $\Delta = \Omega(\log n/n)$ between the real preimage-entropy and the average accessible max-preimage-entropy.

The second part of the construction starts with any function with some noticeable gap Δ and shows how to obtain a UOWHF. This is achieved using the following steps:

1. Gap amplification and transformation of an average type gap into an absolute type of gap.
2. Entropy reduction.
3. Output length reduction.
4. Random inputs collision-resistance to a UOWHF.
5. Removing the non-uniformity.

The composition of Steps 2 through 5 of their construction[2] is summarized in the following theorem, which we later use in a black-box manner:

Theorem 1.

1. *There exists an explicit black-box construction taking parameters a function* $\psi = \{\psi_n\}_{n\in\mathcal{N}}$, *where* $\psi_n : \{0,1\}^{\lambda(n)} \to \{0,1\}^{m(n)}$, *and a number* $\tau = \tau(n)$ *such that if* $H^{\text{eff}}_{p,\text{avg}-\max}(\psi_n) + \omega(\log(n)) \leq \tau(n) \leq H_p(\psi_n)$ *holds, the construction implements a UOWHF with output length and key length* $O(\lambda(n))$.
2. *Moreover, for all efficiently computable* $l = l(n)$ *there exists an explicit black-box construction taking parameters* ψ *(as before) and sets of numbers* $\tau = \tau(n) = \{\tau_{n,i}\}_{i=1}^{l(n)}$, *such that if one of* $\{(\psi_n, \tau_{n,i})\}_{i=1}^{l(n)}$ *satisfies the condition of part (1), the construction implements a UOWHF with output length* $O(\lambda(n) \cdot l(n))$ *and key length of* $O(\lambda(n) \cdot l(n) \cdot \log(l(n)))$.

3 UOWHF from a 2^r-Regular OWF

Let $f : \{0,1\}^n \to \{0,1\}^m$ be a 2^r-regular one-way function. Our construction also works in two steps: First we obtain an entropy gap of $O(\log(n))$ applying f only once and use a variant of the Naor-Yung construction. Next, we show that the type of gap that we get by our first step is almost of the required absolute type. Namely, the average entropy gap is essentially concentrated on a smaller interval of size almost $O(\log(n))$, and in this case the structured gap can be transformed to an absolute type of gap via taking only a super-logarithmic number of independent samples. The main result of this section is:

Theorem 2. *Let* $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ *be a* 2^r*-regular one-way function, where* $r = r(n)$ *is efficiently computable. Then there exists an explicit black-box construction of a universal one-way hash function based on* f *with output length and key length of* $O(n \cdot \log(n) \cdot \alpha(n))$ *for any super-constant function* $\alpha(n)$.

[2] Lemmas 5.3 – 5.4, 5.6 in [2].

3.1 Inaccessible Entropy from 2^r-Regular One-Way Functions

In this case f has exactly 2^{n-r} different images. If we randomly distribute the images among b buckets, we expect to have roughly $\frac{2^{n-r}}{b}$ images in each bucket. Consider the composed function, $F(x,g) = (g(f(x)), g)$ where g is the description of a three-wise independent hash-function from some family $\mathcal{G}$. We show that an appropriate choice of the family $\mathcal{G}$ allows us to reduce the preimage inaccessibility of F to the hardness of the underlying function f.

For injective one-way functions this was already observed in [6]. The difference is that for an injective f, the resulting F is already a universal one-way hash function, whereas in the case where f is regular we get:

Lemma 2. *Let $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ be a 2^r-regular one-way function, where $r = r(n)$ is efficiently computable. Let $d > 0$ and let $\mathcal{G} = \mathcal{G}(n) \stackrel{\text{def}}{=} \mathcal{G}_n^{(n-r)-4d\log(n)}$ be a family of constructible three-wise independent hash functions. Then the function $F : \{0,1\}^n \times \mathcal{G} \to \{0,1\}^{(n-r)-4d\log(n)} \times \mathcal{G}$ given by: $F(x,g) = (g(f(x)), g)$ satisfies the following properties:*

1. $H_p(F) \geq r + 3d\log(n)$.
2. $H^{\text{eff}}_{p,\max}(F) \leq r$.

Proof.

1. Recall that when the input is chosen uniformly at random from the input space the preimage-entropy of F is just the expected log-value of the size of the preimage set. We first compute the expected number of preimages for a fixed x with some random g from $\mathcal{G}$. Since $F(x,g)$ already determines g it follows that any potential preimage must have the same g component.
   For all fixed x, g we have the set equality:

$$F^{-1}(F(x,g)) = \bigcup_{y' : g(f(x)) = g(y')} \left(f^{-1}(y') \times \{g\}\right) \tag{4}$$

   and the union is over disjoint sets. We get that:

$$\begin{aligned} \pi_F(x,g) &= \pi_f(x) + \sum_{y' \neq f(x)} \mathbf{1}_{g(f(x))=g(y')} \cdot |f^{-1}(y')| \\ &= 2^r \cdot \left(1 + \sum_{y' \neq f(x)} \mathbf{1}_{g(f(x))=g(y')}\right) \ , \end{aligned} \tag{5}$$

   where due to the regularity it holds that $\pi_f(x) = |f^{-1}(y)| = 2^r$.
   Now we observe that for every fixed $x \in \{0,1\}^n$ and $y' \neq f(x)$

$$\mathop{\mathbf{E}}_{g \stackrel{r}{\leftarrow} \mathcal{G}} \left[\mathbf{1}_{g(f(x))=g(y')}\right] = 2^{-(n-r-4d\log(n))} \ , \tag{6}$$

   where the equality is due to due to the pair-wise independence of $\mathcal{G}$.

Using (5), (6) and linearity of expectation we have that

$$\mathop{\mathbf{E}}_{g\xleftarrow{r}\mathcal{G}}[\pi_F(x,g)] = 2^r \cdot \left(1 + \sum_{y'\neq f(x)} \mathop{\mathbf{E}}_{g\xleftarrow{r}\mathcal{G}}[\mathbb{1}_{g(f(x))=g(y')}]\right) > 2^{r+4d\cdot\log(n)} \quad , \quad (7)$$

where again due to the regulariry the summation is over $2^{n-r}-1$ indicators. Furthermore, as the family is three-wise independent, we also have that for different $f(x), y', y''$ it holds that the random variables $\mathbb{1}_{g(f(x))=g(y')}$ and $\mathbb{1}_{g(f(x))=g(y'')}$ are independent (and in particular, uncorrelated) and therefore

$$\mathop{\mathbf{V}}_{g\xleftarrow{r}\mathcal{G}}[\pi_F(x,g)] = (2^r)^2 \cdot \sum_{y'\neq f(x)} \mathop{\mathbf{V}}_{g\xleftarrow{r}\mathcal{G}}\left[\mathbb{1}_{g(f(x))=g(y')}\right] \leq \left(2^{r+2\cdot d\cdot\log(n)}\right)^2 \quad . \quad (8)$$

where the equality holds for the sum of uncorrelated random variables and the the inequality holds as for all indicator random variables $\mathbf{V}[\mathbb{1}_A] \leq \mathbf{E}[\mathbb{1}_A]$ and using (6). Now, the Chebysev Inequality establishes that for all $\alpha > 0$:

$$\Pr_{g\xleftarrow{r}\mathcal{G}}\left[\left|\pi_F(x,g) - \mathop{\mathbf{E}}_{g\xleftarrow{r}\mathcal{G}}[\pi_F(x,g)]\right| > \alpha\cdot 2^{r+2\cdot d\cdot\log(n)}\right] < \frac{1}{\alpha^2} \quad . \quad (9)$$

Whenever the event in (9) does not happen plugging (7) we obtain

$$\pi_F(x,G) \geq 2^{r+4\cdot d\cdot\log(n)} - \alpha\cdot 2^{r+2\cdot d\cdot\log(n)} \geq 2^{r+3.5\cdot d\cdot\log(n)} \quad ,$$

for all fixed α and sufficiently large n.
Finally, recall that due to the regularity we always have $\pi_F(x,g) \geq 2^r$ and so using conditional expectation on the event from (9) with $\alpha = 5$ and plugging (7) we obtain:

$$\mathop{\mathbf{E}}_{g\xleftarrow{r}\mathcal{G}}[\log(\pi_F(x,g))] > r + \frac{24}{25}\cdot(3.5\cdot d\cdot\log(n)) \geq r + 3\cdot d\cdot\log(n) \quad . \quad (10)$$

As this holds for every fixed x, it also holds for a random one, and we are done.

2. We show that any efficient algorithm that finds a collision for a random input (X,G) outside of $f^{-1}(X)\times\{G\}$ leads to one that inverts f. Let A_F be an F-collision-finder. We denote the randomness taken by A_F explicitly (as an additional argument) by r, and for some fixed randomness r and a fixed input x, denote by $\epsilon_{x,r} \stackrel{\text{def}}{=} \Pr_{g\xleftarrow{r}\mathcal{G}}[A_F(x,g,r) \notin f^{-1}(f(x))\times\{g\}]$.
We show[3] that the algorithm B inverts $y = f(x')$, where $x' \xleftarrow{r} \{0,1\}^n$ uniformly at random, with probability at least ϵ/n^{-d}.

[3] In similar manner to [6] and [2].

Algorithm $B(y)$ // On input $y \in \{0,1\}^n$

(a) Choose x uniformly at random from $\{0,1\}^n$.
(b) Choose randomness r for A_F uniformly at random.
(c) Choose g uniformly at random subject to $g(f(x)) = g(y)$.
(d) Output $\phi_1(A_F(x,g,r))$.

It holds that

$$\epsilon_{x,r} = \sum_{y \neq f(x)} \Pr_{g \xleftarrow{r} \mathcal{G}} \left[\phi_1(A_F(x,g,r)) \in f^{-1}(y)\right] \tag{11}$$

$$= \sum_{y \neq f(x)} \Pr_{g \xleftarrow{r} \mathcal{G}} \left[\phi_1(A_F(x,g,r)) \in f^{-1}(y) \mid g(f(x)) = g(y)\right] \tag{12}$$

$$\cdot \Pr_{g \xleftarrow{r} \mathcal{G}} \left[g(f(x)) = g(y)\right]$$

$$= \sum_{y \neq f(x)} \Pr_{g \xleftarrow{r} \mathcal{G}} \left[\phi_1(A_F(x,g,r)) \in f^{-1}(y) \mid g(f(x)) = g(y)\right] \tag{13}$$

$$\cdot\, 2^{-(n-r-d\log(n))} \ .$$

It follows that conditioned on the random choices $X = x$ and $R = r$ of the algorithm B, we obtain that

$$\begin{aligned}
&\Pr_{x' \xleftarrow{r} \{0,1\}^n} [B(f(x')) \in f^{-1}(f(x')) \mid X = x \wedge R = r] \\
&= \Pr_{y \xleftarrow{r} f(\{0,1\}^n)} [B(y) \in f^{-1}(y) \mid X = x \wedge R = r] \\
&\geq \sum_{y \neq f(x)} \Pr[Y = y] \cdot \Pr_{g \xleftarrow{r} \mathcal{G}} \left[\phi_1(A_F(x,g,r)) \in f^{-1}(y) \mid g(f(x)) = g(y)\right] \\
&\geq 2^{-d\log(n)} \cdot \epsilon_{x,r} \ .
\end{aligned} \tag{14}$$

As X and R are chosen uniformly at random, and by the fact that $\epsilon = \mathbf{E}[\epsilon_{X,R}]$ we get that the algorithm inverts f with probability at least $n^{-d} \cdot \epsilon$. Thus we have shown that A_F's output on input (x,g) is limited to $f^{-1}(f(x)) \times \{g\}$. By the regularity of f we have that its size is exactly 2^r. Thus F has accessible max-preimage-entropy at most r.

□

We next show that by a more careful analysis of the amplification results in an almost-linear construction.

3.2 Amplifying the Entropy Gap and Converting Average to Absolute Entropy Gaps

Lemma 3 (Fast gap amplification and real- to min- preimage-entropy conversion). *Let f and F be as in Lemma 2, F^t be the t-fold application of F and $\alpha(n)$ be any*

super-constant function. Then for $t = \alpha(n) \cdot \log(n)$, F^t has a strong inaccessible entropy gap of $\alpha(n) \cdot \log^2(n)$. Moreover, the following entropy-gap holds:

1. $H_{p,\min}(F^t) \geq t(r + 2d\log(n))$.
2. $H^{\text{eff}}_{p,\max}(F^t) \leq t \cdot r$.

Proof.

1. We first show that by the Markov inequality, the probability that the point-wise entropy exceeds its expected value by more than $\alpha(n) \cdot \log(n)$ bits is negligible. Specifically, from the first part of Lemma 2, we know that the expected value of the preimage size of inputs to F is $2^{r+4d\log(n)}$. The Markov inequality asserts that the probability we get an input with more than $2^{\alpha(n)\cdot\log(n)} \cdot 2^{r+4d\log(n)}$ preimages is at most $1/n^{\alpha(n)}$. Let us denote this 'bad' event as A. By Lemma 5 [4] we get that $\mathbf{E}[\log(\pi_F(X,G)) \mid \overline{A}] \geq r + 4d\log(n) - n^{1-\alpha(n)}$.

   From now on we assume that this unlikely event does not happen. We get that the value of the real entropy is limited to an interval of size $O(\alpha(n) \cdot \log(n))$, as we always have a at least r bits of point-wise entropy due to the regularity of f.

   Now we can apply the Hoeffding bound which asserts that for this case a super-logarithmic number of repetitions suffice to bound from below the min-preimage-entropy of the t-fold application of F. Specifically, we get that

$$\Pr_{(x,g)^t \xleftarrow{r} (\{0,1\}^n \times \mathcal{G})^t} \left[\log\left(\pi_{F^t}((x,g)^t)\right) \geq t(r + 2d\log(n))\right] \leq \exp\left(\frac{-2t}{\alpha^2(n)}\right).$$

   The choice of $t = O(\alpha^3(n) \cdot \log(n))$ ensures that this happens with probability at most $1/n^{\alpha(n)}$.
2. This is just the t-fold accessible max-preimage-entropy we get by the second part of Lemma 2 and Corollary 1.

□

Proof (Theorem 2). By Lemma 3, F^t already has the required strong type of entropy gap between its accessible max-preimage-entropy and its real min-preimage-entropy. Moreover, it tells us exactly where this gap is (there are at most $t \cdot r$ bits of accessible max-preimage-entropy). Now, note that if $\alpha(n)$ is a super-constant function, then so is $\alpha'(n) = \alpha^{1/3}(n)$. Finally, utilizing Theorem 1 with parameter $(F^t, t \cdot (r + 2d\log(n)))$ completes the construction and yields a UOWHF with output length and key length $O(n \cdot \log(n) \cdot \alpha(n))$. □

[4] We use Lemma 5 in the uniform setting, where $l = l(n) = n$ (as we consider the point-wise Shannon entropy) and A_n is an event that happens with some negligible probability, that is, $\Pr[A_n] < n^{-\alpha(n)}$ for some super-constant function $\alpha(n)$.

4 UOWHF from a $(2^{r(n)}, 2^{s(n)})$-Roughly-Regular OWF

The main theorem proved in this section is:

Theorem 3. *Let $f : \{0,1\}^n \to \{0,1\}^{m(n)}$ be a $(2^r, 2^s)$-roughly-regular one-way function, where $r = r(n)$ and $s = s(n)$ are efficiently computable. Then there exists an explicit construction of a UOWHF with output length and key length of $\tilde{O}(n \cdot s^6(n))$ (resp., $\tilde{O}(n \cdot s^4(n))$) in the uniform (resp., non-uniform) model.*

4.1 $\log(n)/s(n)$ Bits of Average Inaccessible Entropy

Haitner et. al. showed that for a general one-way function f, a random truncation of a hashing of $f(x)$ using a three-wise independent family of hash functions yields an average entropy gap of $\Omega(\log(n)/n)$ entropy bits. We observe that a modification of their first step achieves an average inaccessible entropy gap of $\log(n)/s(n)$ bits from any $(2^{r(n)}, 2^{s(n)})$-roughly-regular one-way function.

The idea is to divide the images $f(x)$ (and respectively, the inputs x) into buckets, such that every bucket contains images with roughly the same number of preimages. We set $m \stackrel{\text{def}}{=} s(n)/d\log(n)$ and $J \stackrel{\text{def}}{=} \{j_0, \ldots, j_{m-1}\}$, where $j_i \stackrel{\text{def}}{=} n - r(n) - s(n) + (i-1)d\log(n)$, and show that truncating the output of the application of a three-wise independent hashing of $f(x)$ to a random length from J yields a function with the required gap. Recall that $H_{f(X)}(f(x)) \in (j_i, j_{i+1}]$ if and only if $\pi_f(x) \in [2^{r+s-i(d\log(n))}, 2^{r+s-(i-1)(d\log(n))})$. Let us denote $q_i \stackrel{\text{def}}{=} \Pr[H_{f(X)}(f(x)) \in (j_i, j_{i+1}]]$. By the roughly-regularity assumption on f, it holds that $\sum_{i=1}^{m} q_j = 1$.Now we set $\mathcal{G} \stackrel{\text{def}}{=} \mathcal{G}_n^n$ a family of three-wise independent hash functions, $X \stackrel{\text{def}}{=} \{0,1\}^n$ and define $F : X \times \mathcal{G} \times J \to X \times \mathcal{G} \times J$ as $F(x,g,j) = (g(f(x))_{1,\ldots,j} \| 0^{n-j}, g, j)$, where we denote the domain and range of F by $\mathcal{Z} \stackrel{\text{def}}{=} X \times \mathcal{G} \times J$.

Lemma 4. *The function F as defined above has an average preimage-entropy gap of $s(n)/log(n)$ bits.*

Proof. Recall that our goal in this step is to achieve an average inaccessible entropy gap of $\Omega(\log(n)/s(n))$ bits. That is, we need to show that for each $z = (x, g, j_i)$ there exists a set S_z, such that: (1) any efficient collision-finder outputs an element of S_z (except for an event that happens with negligible probability) and (2) $\mathbf{E}_{z \xleftarrow{r} \mathcal{Z}}[\log(\pi_F(z)) - \log(|S_z|)] > \Omega(\log(n)/s(n))$.

In a similar manner to the regular case, the set of inputs accessible by an efficient algorithm is limited only to those with relatively few images, where "few" corresponds to the length of the random truncation. Essentially, we show that when we hash to length j_i, any preimages an efficient algorithm finds are either already preimages of $f(x)$ (we refer to these as 'trivial' collisions) [5] or stem from some non-trivial collision, that is $F(x', g, j_i) = F(x, g, j_i)$ but $f(x) \neq f(x')$.

[5] Note that the definition of a one-way function does not rule out the possibility that given a preimage it is difficult to compute other preimages from $f^{-1}(f(x))$.

For the latter, we further distinguish between those x that have significantly fewer preimages than expected for a random function with output length j_i, and the rest. More precisely, we consider those preimages $z' = (x', g, j_i)$ for which $\pi_f(x') \leq 2^{j_{i+2}}$ and call these 'j_{i+2}-light' preimages of $f(x)$. The remaining 'heavy' collisions stem from inputs z' for which $\pi_f(x') > 2^{j_{i+2}}$.

We define:

$$T_z \stackrel{\text{def}}{=} T_{(x,g,j_i)} = f^{-1}(f(x)) \times \{g\} \times \{j\},$$

$$L_z \stackrel{\text{def}}{=} L_{(x,g,j_i)} = \{x' \in \{0,1\}^n \mid g(f(x))_{1,\ldots,j_i} = g(f(x'))_{1,\ldots,j_i} \\ \wedge H_{f(X)}(f(x)) \geq j_{i+2} \wedge x' \notin f^{-1}(f(x))\} \times \{g\} \times \{j\}$$

and

$$H_z \stackrel{\text{def}}{=} H_{(x,g,j_i)} = \{x' \in \{0,1\}^n | g(f(x))_{1,\ldots,j_i} = g(f(x'))_{1,\ldots,j_i} \\ \wedge H_{f(X)}(f(x)) < j_{i+2} \wedge x' \notin f^{-1}(f(x))\} \times \{g\} \times \{j\} \ ,$$

where T, L and H stand for 'trivial', 'light' and 'heavy', respectively. It follows that for every z,

$$F^{-1}(F(z)) = T_z \cup L_z \cup H_z \ , \tag{15}$$

where the union is over disjoint sets.

The rest of the proof is involved with proving that indeed the only accessible sets to any efficient algorithm are $T_z \cup L_z$, and that they constitute a large fraction of the preimage set $F^{-1}(F(z))$. The analysis follows the construction from [3] and is brought for completeness in Appendix B. □

4.2 Faster Amplification of the Inaccessible Entropy Gap of F

Our goal in this section is to amplify the entropy gap of F from the previous section. We show how to construct a function F' with $\omega(\log(n))$ bits of inaccessible entropy with an absolute type of gap.

Haitner et. al. [2] assert that independent repetitions of F achieve both these goals. They show that $\tilde{O}(n^4)$ repetitions are enough for getting this gap from an arbitrary one-way function. We are able to utilize the information about the underlying f (and in turn, that of F) and get a faster convergence, using the roughly-regularity assumption.

Set $\Delta \stackrel{\text{def}}{=} (c \cdot \log(n)/s(n))$ as the entropy gap of F, where c is the constant corresponding to the Ω notation, and fix k, such that F has preimage-entropy $H_p(F) = k + \Delta$. Lemma 4 asserts that $H^{\text{eff}}_{p,\text{avg}-\max}(F) \leq k$. Using (3) and Corollary 1 we know that for the t-fold parallel repetition of F it holds that

$$H_p(F^t) = t \cdot (k + \Delta) \ , \tag{16}$$

$$H^{\text{eff}}_{p,\text{avg}-\max}(F^t) \leq t \cdot k \ . \tag{17}$$

Thus for F^t we obtain an average entropy gap of $t \cdot \Delta$ bits.

Using the analysis of Lemma 4 and Lemma 1 we get that for an input $z^t = (z_1, \ldots, z_t)$ to F^t, the only accessible inputs to F^t are those that are contained in $S_z = (T_{z_1} \cup L_{z_1}) \times \cdots \times (T_{z_t} \cup L_{z_t})$, and that the set of preimages of z^t is just $F^{t^{-1}}(F^t(z^t)) = (T_{z_1} \cup L_{z_1} \cup H_{z_1}) \times \cdots \times (T_{z_t} \cup L_{z_t} \cup H_{z_t})$, except for an event B_1 that occurs with negligible probability. Next, we would like to apply the Hoeffding bound to get the required gap. Similarly to Lemma 3 we show that although for some inputs the preimage size of F may be very large (a priori there may be inputs with up to 2^n preimages, but not more, since $F(x, g, j_i)$ determines (g, j_i) uniquely as part of its output), this is not likely. First observe that $\log(\pi_F(z)) \in [r, n]$ for all z. This is due to the fact that every image of $f(x)$ has at least $2^{r(n)}$ preimages. We show that we can bound this also from above: except with negligible probability we have that for any super-constant function $\alpha(n)$: $\log(|T_z \cup L_z|) \leq \log(\pi_F(z)) < r(n) + s(n) + d\log(n) + \alpha(n)\log(n)$. Consider $\pi_F(Z)$ for a uniformly chosen random input $Z = (X, G, J)$. This value is maximized for $J = j_0$ because of the inclusion $\phi_1(F^{-1}(F(x, g, j_i'))) \subset \phi_1(F^{-1}(F(x, g, j_i)))$ for $j_i \leq j_i'$. It follows that in order to bound $\mathbf{E}[\pi_F(X, G, J)]$ it is sufficient to bound $\mathbf{E}[\pi_F(X, G, j_0)]$.

As in Lemma 2, using the three-wise independence of $\mathcal{G}$, and the roughly-regularity of f we have that for fixed x it holds that:

$$\mathop{\mathbf{E}}_{g \xleftarrow{r} \mathcal{G}} [\pi_F(x, g, j_0)] \leq 2^{r(n)+s(n)+d\log(n)+2} .$$

Next, fix any super-constant function $\alpha(n)$. Markov's inequality asserts that

$$\Pr_{g \xleftarrow{r} \mathcal{G}} \left[\pi_F(x, g, j_0) \geq 2^{r(n)+s(n)+d\log(n)} \cdot 2^{\alpha(n)\log(n)}\right] \leq n^{-\alpha(n)} .$$

Denote the event that this happens in any of the repetitions by B_2 and note that it happens only with negligible probability (as t is polynomial in n and using the union bound). We summarize this as follows: whenever B_2 does not occur, we get that

$$r(n) \leq \log(|T_Z \cup L_Z|) \leq \log(\pi_F(Z)) \leq r(n) + s(n) + (d + \alpha(n))\log(n) .$$

When this is the case, both quantities are within an interval of size $s' \stackrel{\text{def}}{=} 3 \cdot \max\{s(n), \alpha(n) \cdot \log(n)\}$.

By Lemma 5 we know that the preimage-entropy and the average accessible max-preimage-entropy values change by at most a negligible quantity when ignoring an event of negligible probability. Specifically, we get that whenever $\overline{B_1} \wedge \overline{B_2}$ happen we have:

$$k' \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}} \left[\log(|S_z|) \,|\, \overline{B_1} \wedge \overline{B_2}\right] \leq \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}} [\log(|S_z|)] + \mathsf{negl}(n) \tag{18}$$

and

$$k'' \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}} \left[\log(\pi_F(z)) \,|\, \overline{B_1} \wedge \overline{B_2}\right] \geq \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}} [\log(\pi_F(z))] - \mathsf{negl}(n) \tag{19}$$

with a gap of $\Delta' \stackrel{\text{def}}{=} k'' - k' \geq \Delta - \mathsf{negl}(n)$.

The Hoeffding bound yields that setting $t \stackrel{\text{def}}{=} O(\frac{s'^2(n)\cdot s^2(n)}{\log(n)})$ assures that the inaccuracies due to the sampling of the independent inputs to F are already smaller than the accumulated gap. Specifically:

$$\Pr_{z^t \xleftarrow{r} \mathcal{Z}^t}\left[\log\left(|S_{z^t}|\right) > t\cdot k' + \frac{c}{6}\cdot s'(n)\cdot\sqrt{t\cdot\alpha(n)\cdot\log(n)}\right] \leq n^{-\alpha(n)} \ , \tag{20}$$

$$\Pr_{z^t \xleftarrow{r} \mathcal{Z}^t}\left[\log\left(\pi_{F^t}(z^t)\right) < t\cdot(k'+\Delta')) - \frac{c}{6}\cdot s'(n)\cdot\sqrt{t\cdot\alpha(n)\cdot\log(n)}\right] \leq n^{-\alpha(n)}. \tag{21}$$

Plugging (18) and (19) we get that except with negligible probability there is an absolute entropy gap of at least

$$t\cdot\Delta - t\cdot\mathsf{negl}(n) - \frac{c}{3}\cdot s'(n)\cdot\sqrt{t\cdot\alpha(n)\cdot\log(n)} \in \omega(\log(n)) \ . \tag{22}$$

4.3 A UOWHF in the Non-uniform Model

To finish the construction we would like to apply the first part of Theorem 1. We use the preimage-entropy of F from Section 4.1 (in the form of a non-uniform advice), which equals $k+\Delta$. By what we have shown in Section 4.2 it holds that F^t has real min-preimage-entropy of at least $\tau \stackrel{\text{def}}{=} t\cdot(k+\Delta) - \frac{c}{4}\cdot s'(n)\cdot\sqrt{t\cdot\alpha(n)\cdot\log(n)}$ bits. Additionally, it enjoys the required absolute entropy gap. The first part of Theorem 1 with parameters (F^t,τ) yields a UOWHF with output length and key length $O(n\cdot s'^2(n)\cdot s^2(n)/\log(n))$.

4.4 An Efficient Non-uniform to Uniform Reduction

As explained, the construction obtained requires a non-uniform advice (i.e., the Shannon preimage-entropy of f). We remove the non-uniformity by 'trying all possibilities'. However, as opposed to the case of a general one-way function, where we need to try $O(n^2)$ different values, we show that using the roughness regularity assumption we only need $O(s^2(n))$ tries.

Recall that by the roughly-regularity assumption on f, it follows that the preimage-entropy of F lies in the interval $[r+d\log(n), s+d\log(n)]$.

For $i \in [\lceil 4\cdot s(n)/c\log(n)\rceil]$ set $k_i \stackrel{\text{def}}{=} r+d\log(n)+i\cdot\frac{1}{4\cdot s(n)}$. It holds that one of the k_i is within an additive distance of $\frac{\Delta}{4}$ from the real value $k+\Delta$. Accordingly, set $\tau_i \stackrel{\text{def}}{=} t\cdot(k_i+\Delta) - \frac{c}{4}\cdot s'(n)\cdot\sqrt{t\cdot\alpha(n)\cdot\log(n)}$. It follows that for the same i, (F_t,τ_i) satisfies the premise of the first part of Theorem 1, and thus the second part of the theorem yields a construction of a UOWHF with output length of $O(n\cdot s'^2(n)\cdot s^4(n)/\log^3(n))$ and key length of $O(n\cdot s'^2(n)\cdot s^4(n)/\log^2(n))$.

5 Conclusions

We demonstrated how to obtain more efficient constructions of a UOWHF from different assumptions on the structure of the underlying OWF. For the case of known regularity the resulting construction is very efficient and makes an almost logarithmic number of calls to the underlying OWF. In the case when the underlying OWF is known to be either 2^{r_1}-regular or 2^{r_2}-regular (i.e., the construction is given r_1 and r_2 and should be secure when instantiated with any function of the corresponding regularity), we observe that one obtains a construction that makes $\tilde{O}(n)$ calls (combining our construction for the regular case with the second part of Theorem 1). Of course, the main open problem remains to further improve the construction of Haitner et. al. for a general OWF.

Acknowledgments. We would like to thank the anonymous reviewers for their helpful comments. The first author would like to thank David Adjiashvili and Sandro Coretti for their comments on an earlier version of this work.

References

1. Goldreich, O., Krawczyk, H., Luby, M.: On the existence of pseudorandom generators. SIAM J. Comput. 22(6), 1163–1175 (1993)
2. Haitner, I., Holenstein, T., Reingold, O., Vadhan, S., Wee, H.: Universal One-Way Hash Functions via Inaccessible Entropy. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 616–637. Springer, Heidelberg (2010)
3. Haitner, I., Nguyen, M.-H., Ong, S.J., Reingold, O., Vadhan, S.P.: Statistically hiding commitments and statistical zero-knowledge arguments from any one-way function. SIAM J. Comput. 39(3), 1153–1218 (2009)
4. Haitner, I., Reingold, O., Vadhan, S.P.: Efficiency improvements in constructing pseudorandom generators from one-way functions. In: Schulman, L.J. (ed.) STOC, pp. 437–446. ACM (2010)
5. Haitner, I., Reingold, O., Vadhan, S.P., Wee, H.: Inaccessible entropy. In: Mitzenmacher, M. (ed.) STOC, pp. 611–620. ACM (2009)
6. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: STOC, pp. 33–43. ACM (1989)
7. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: STOC, pp. 387–394. ACM (1990)
8. De Santis, A., Yung, M.: On the Design of Provably-Secure Cryptographic Hash Functions. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 412–431. Springer, Heidelberg (1991)

A Further Preliminaries

A.1 The Hoeffding Bound

For independent bounded random variables $X_1, \ldots, X_t$, where $X_i \in [a_i, b_i]$, set $S_t = \Sigma_{i=1}^{t} X_i$, then:

$$\Pr\left[|S_t - \mathbf{E}[S_t]| \geq k\right] \leq 2 \cdot \exp\left(\frac{-2k^2}{\Sigma_{i=1}^{t}(b_i - a_i)^2}\right) .$$

A.2 *t*-wise Independent Hashing

Let $\mathcal{G}_n^m \stackrel{\text{def}}{=} \{g_k : \{0,1\}^n \to \{0,1\}^m\}_{k \in K}$ be a family of keyed functions. $\mathcal{G}_n^m$ is t-wise independent if for all $y_1, \ldots, y_t \in \{0,1\}^m$ and all distinct $x_1, \ldots, x_t \in \{0,1\}^n$ it holds that

$$\Pr_{k \stackrel{r}{\leftarrow} K}\left[g_k(x_1) = y_1 \wedge \cdots \wedge g_k(x_t) = y_t\right] = 2^{-tm} \ .$$

The family is called constructible if, for all $y_1, \ldots, y_s$ and distinct $x_1, \ldots, x_s$ where $s \leq t$, it is possible to sample a function uniformly subject to $g_K(x_1) = y_1, \ldots, g_K(x_s) = y_s$.

It is well-known that if $\mathcal{G}_n^n$ is a t-wise independent family, then by truncating the last $n - l$ bits of $\mathcal{G}_n^n$ one gets a t-wise independent family $\mathcal{G}_n^l$. Moreover, if $\mathcal{G}_n^n$ is constructible, then so is $\mathcal{G}_n^l$.

The next lemma shows that ignoring an unlikely event of a random variable that takes a value in some limited range, does not change much its expected value. The standard proof is omitted in this extended abstract.

Lemma 5. *Let X be a random variable with $\mathrm{Supp}(X) \subset [0, l]$ and A an event that happens with probability at most ϵ. Then:*

$$\left|\mathbf{E}\left[X\right] - \mathbf{E}\left[X|\overline{A}\right]\right| \leq 2 \cdot l \cdot \epsilon \ . \tag{23}$$

B Proof of Lemma 4, continued.

Our next goal is to show that the sets $\{T_z \cup L_z\}_{z \in Z}$ satisfy the needed requirements. Claim 4.9 in [2] shows that any efficient collision-finder cannot (except with negligible probability) output a preimage of $F(z)$ in H_z, as such an algorithm can be used to invert f. Specifically, they show (again, using the constructibility of the three-wise independent hash family as in the second part of Lemma 2) how to efficiently convert any F-collision-finder that outputs a preimage from H_z with probability ϵ to one that inverts a random input of f with probability ϵ/n^d.

As the preimage sets $\{H_z\}_{z \in \mathcal{Z}}$ are inaccessible, it remains to show that they constitute a noticeable part of the preimage sets. In order to complete the proof, we need to bound:

$$\begin{aligned}
\mathop{\mathbf{E}}_{z \stackrel{r}{\leftarrow} \mathcal{Z}}\left[\pi_F(z)\right] - \mathop{\mathbf{E}}_{z \stackrel{r}{\leftarrow} \mathcal{Z}}\left[\log(|T_z \cup L_z|)\right] &= \mathop{\mathbf{E}}_{z \stackrel{r}{\leftarrow} \mathcal{Z}}\left[\log\left(\frac{\pi_F(z)}{|T_z \cup L_z|}\right)\right] && (24)\\
&= \mathop{\mathbf{E}}_{z \stackrel{r}{\leftarrow} \mathcal{Z}}\left[\log\left(\frac{|T_z| + |L_z| + |H_z|}{|T_z| + |L_z|}\right)\right] && (25)\\
&= \mathop{\mathbf{E}}_{z \stackrel{r}{\leftarrow} \mathcal{Z}}\left[\log\left(1 + \frac{|H_z|}{|T_z| + |L_z|}\right)\right] && (26)\\
&\geq \frac{1}{2} \mathop{\mathbf{E}}_{z \stackrel{r}{\leftarrow} \mathcal{Z}}\left[\frac{|H_z|}{|T_z| + |L_z| + |H_z|}\right] && (27)
\end{aligned}$$

where the second equality is due to the partition in (15) and the inequality uses the fact that $\log(1+x) \geq x/2$ for $x \in [0,1)$. Thus, it is left to show that indeed $|H_z|$ constitutes a noticeable part of $\pi_F(z)$.

Proposition 1. *Conditioned on $X = x$ and $J = j_i$, define the events:*

$$E^1_{j_i} \stackrel{\text{def}}{=} \left\{|H_z| + |L_z| \leq 3 \cdot 2^{n-j_i}\right\}$$
$$E^2_{j_i} \stackrel{\text{def}}{=} \left\{|H_z| \geq \left(q_i - 4 \cdot \sqrt{1/n^d}\right) \cdot 2^{n-j_i-1}\right\} \ .$$

Then $\Pr\limits_{g \xleftarrow{r} \mathcal{G}}[E^1_{j_i}] > 2/3$ *and* $\Pr\limits_{g \xleftarrow{r} \mathcal{G}}[E^2_{j_i}] > 3/4$ *hold.*

This is just Claim 4.11 from [2] [6]. It follows that:

$$= \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}}[\log(\pi_F(z))] - \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}}[\log(|T_z \cup L_z|)] \tag{28}$$

$$\geq \frac{1}{2} \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}}\left[\frac{|H_z|}{|T_z| + |L_z| + |H_z|}\right] \tag{29}$$

$$= \frac{1}{2} \sum_{i=0}^{m-1} \Pr_{z \xleftarrow{r} \mathcal{Z}}[J = j_i] \cdot \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}}\left[\frac{|H_z|}{|T_z| + |L_z| + |H_z|} \;\middle|\; J = j_i\right] \tag{30}$$

$$\geq \frac{1}{2m} \sum_{i=0}^{m-1} \Pr_{z \xleftarrow{r} \mathcal{Z}}[H_{f(X)}(f(X)) > j_i] \cdot \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}}\left[\frac{|H_z|}{|T_z| + |L_z| + |H_z|} \;\middle|\; H_{f(X)}(f(X)) > j_i, J = j_i\right] \tag{31}$$

$$\geq \frac{1}{2m} \sum_{i=0}^{m-1} (q_i + \cdots + q_m) \cdot (1 - \frac{1}{3} - \frac{1}{4}) \cdot \mathop{\mathbf{E}}_{z \xleftarrow{r} \mathcal{Z}}\left[\frac{|H_z|}{|T_z| + |L_z| + |H_z|} \;\middle|\; E^1_{j_i}, E^2_{j_i}, H_{f(X)}(f(X)) > j_i, J = j_i\right] \tag{32}$$

$$\geq \frac{1}{2m} \sum_{i=0}^{m-1} (q_i + \cdots + q_m) \cdot (1 - \frac{1}{3} - \frac{1}{4}) \cdot \frac{(q_{i+1} - 4/(n^{d/2})) \cdot 2^{n-j_i-1}}{2^{n-j_i+2}} \tag{33}$$

$$\geq \frac{1}{96m} \sum_{0 \leq i \leq k \leq m-1} (q_i \cdot q_k) - O\left(\frac{1}{n^{d/2}}\right) \tag{34}$$

$$\geq \frac{1}{200m} - \mathsf{negl}(n) - O(n^{-d/2}) \ , \tag{35}$$

where we used conditional expectations, the union bound, the fact that $H_{f(X)}(f(x)) \geq j_i$ is equivalent to $|T_z| \leq 2^{n-j_i}$ and the roughness-regularity assumption that $\sum_{i=1}^{m} q_i = 1 - \mathsf{negl}(n)$.

[6] We use it with $\alpha = 4$ and note that $\texttt{Heavy} = |H_z|$ and $\texttt{Light} = |L_z|$.

We conclude that the log-size of the set of the accessible inputs to an efficient collision-finder is, on average, bounded away from the point-wise entropy. Put differently, we get a noticeable fraction of $\Omega(1/m) = \Omega(\log(n)/s(n))$ average inaccessible entropy bits.

Random Mappings with Restricted Preimages

Andrew MacFie and Daniel Panario

School of Mathematics and Statistics
Carleton University, Ottawa, ON K1S 5B6, Canada
andrewmacfie@gmail.com, daniel@math.carleton.ca

Abstract. In this paper we refer to finite endofunctions where each image has at most r preimages as r-mappings. Probabilistic analysis of these mappings provides heuristics for problems arising in cryptography which involve similar but more complicated classes of mappings. We give asymptotic probabilities and expectations related to the joint distribution of some pairs of mapping parameters for these mappings, extending work done previously for general, unrestricted mappings. We give an elementary derivation of the expected value of the maximum of tail length and cycle length of a random node in a random r-mapping, which is a useful component in the heuristic analysis of Pollard's rho algorithm. All distributions considered are uniform.

Keywords: Random mappings, preimages, asymptotic probabilities.

Dedicated to the memory of Philippe Flajolet

1 Introduction

Let $\mathcal{F}_n$ be the set of functions ("mappings") from the set $[1..n]$ to itself. Mappings have a combinatorial structure: With any $\varphi \in \mathcal{F}_n$ there is associated a functional graph on n nodes, with a directed edge from vertex u to vertex v if and only if $\varphi(u) = v$. Clearly, we have $|\mathcal{F}_n| = n^n$. In applications, when the analysis of the mappings in a subset of $\mathcal{F}_n$ is (so far) intractable, they are heuristically modeled by the "average" map from the entire set $\mathcal{F}_n$ if, for example, experimental simulations give justification for this. Thus, once the cardinality of $\mathcal{F}_n$ is obtained, it is natural to seek to count the number of mappings of a certain type and find probabilities. Specifically, one may define functions from the set of mappings to the nonnegative integers, such as the longest possible period of the sequence of function iterates, and look at the distribution of these values. This leads to "global" results about the structure of mappings. "Local" properties have been studied too: One may ask about functions on the set of nodes in mappings, such as the period of the sequence of function iterates starting at a given node.

There is plenty of work along these lines in the literature [1,2,3,4,5,6,7,8,9,10,11]. A clear framework for random mapping problems based on generating functions and asymptotic analysis was laid down in [12] (and later [13]), which guides the present work. An interesting extension to

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 254–270, 2012.

parts of [12] can be found in [14]. Some general tools applicable to random mappings were presented in [15].

One may also work with some subset of $\mathcal{F}_n$ instead of the entire set, as done for example in [16,17]. Flajolet and Odlyzko [12] define an r-node in a functional graph as a node with indegree r. We define an r-mapping as a mapping $\varphi : [1..n] \to [1..n]$ where each node in the functional graph has indegree at most r, i.e.

$$\max_{u\in[1..n]} \left|\varphi^{-1}(u)\right| \leq r.$$

Let $\mathcal{F}_n^r$ be the set of r-mappings in $\mathcal{F}_n$. In this paper we study the functions $\mathcal{F}_n^r$, where $r \geq 2$, considering each to be equally likely.

In some applications, we are concerned with a set of mappings to which $\mathcal{F}_n^r$ gives a "closer" approximation than $\mathcal{F}_n$ since we know the mappings have the bounded-preimage-cardinality property. This is the case, for example, in Pollard's rho algorithm [18] for integer factorization. In the analysis of that algorithm we are given a composite integer m and we are interested in properties of the polynomial mappings $x \mapsto x^2 - 1 \bmod k$, where $k|m$. If k is prime, this mapping has the property that every image has at most 2 preimages. This type of property can also be desirable for mappings used in cryptographic hash functions.

The same kinds of analysis done for general mappings mentioned above have been done for r-mappings [19,20,21,22]. As discussed in [23], largely the same methods used in the general mappings case can be used for r-mappings. This paper continues such work. In Sect. 2 we present the generating function framework for studying r-mappings, which is a slight modification of the scheme used in [12] for general mappings. We begin the analysis in Sect. 3, where we introduce some r-mapping parameters and discuss previous results for distributions of single parameters. In Sect. 4 we look at pairs of parameters and we give some new results, such as number of size-$\mathfrak{r}_1$ cycles and number of size-$\mathfrak{r}_2$ trees. Then, in Sect. 5 we present some results on extremal parameters such as expected largest component size. In Sect. 6 we rederive a known result on the expected value of the maximum of tail length and cycle length of a random node in a random r-mapping using generating functions. We also discuss how this can be used in the analysis of the Pollard-Rho algorithm when Brent's variant for cycle detection is used. We conclude in Sect. 7 by mentioning opportunities for further work.

2 Basics

In this paper we use the symbolic method for generating functions along with singularity analysis to derive our results. These methods are the subject of [13], where they are described in great generality.

We begin, closely following [12], by finding the following labeled combinatorial constructions for the functional graphs corresponding to the mappings $\mathcal{F}_n^r$:

$$\begin{aligned}
&\text{Functional graph:} && \mathcal{F} \cong \textsc{Set}(\mathcal{C}),\\
&\text{Connected component:} && \mathcal{C} \cong \textsc{Cyc}_{\geq 1}(\mathcal{R}),\\
&\text{Root:} && \mathcal{R} \cong \mathcal{N} \star \textsc{Set}_{\leq r-1}(\mathcal{T}),\\
&r\text{-ary tree:} && \mathcal{T} \cong \mathcal{N} \star \textsc{Set}_{\leq r}(\mathcal{T}),\\
&\text{Node:} && \mathcal{N} \cong \{\textcircled{1}\}.
\end{aligned}$$

This says that the functional graphs of r-mappings are sets of connected components; the components are directed cycles of nodes with up to $r-1$ children; and each of those children is the root of a tree in which each node can have up to r children.

According to the symbolic method [13], these constructions immediately translate into the following equations in exponential generating functions:

$$\begin{aligned}
&\text{Functional graph:} && F(z) = \exp(C(z)),\\
&\text{Connected component:} && C(z) = \log\left((1-R(z))^{-1}\right),\\
&\text{Root:} && R(z) = z\sum_{i=0}^{r-1} \tfrac{T(z)^i}{i!},\\
&r\text{-ary tree:} && T(z) = z\sum_{i=0}^{r} \tfrac{T(z)^i}{i!}.
\end{aligned}$$

By exponential generating function we mean, for example, that $C(z) = \sum_{n\geq 0} c_n z^n/n!$, where c_n is the number of connected component objects with n nodes.

In this paper, we apply Flajolet and Odlyzko's singularity analysis [24] repeatedly. We only require and state its simplest form, which can be expressed as the following theorem; for more details see [13,24].

Theorem 1 (Singularity analysis [24]). *Fix the range of* Arg *to be* $[-\pi, \pi)$. *Let* f *be a complex function analytic in a domain*

$$D = \left\{ z : |z| \leq s_1, |\operatorname{Arg}(z-s)| > \frac{\pi}{2} - \eta \right\},$$

where $s, s_1 > s$, *and* η *are three positive real numbers. Assume that, with* $\sigma(u) = u^\alpha \log^\beta u$ *and* $\alpha \notin \{0, -1, -2, \dots\}$, *we have*

$$f(z) \sim \sigma\left(\frac{1}{1-z/s}\right) \qquad \text{as } z \to s \text{ in } D. \tag{1}$$

Then the Maclaurin coefficients of f *satisfy, as* $n \to \infty$,

$$[z^n] f(z) \sim s^{-n} \frac{\sigma(n)}{n\Gamma(\alpha)}. \tag{2}$$

Put simply, singularity analysis entails that if the generating function $f(z)$ of interest behaves as stated in (1) when z is close to the dominant singularity s (the unique singularity of smallest modulus) then, asymptotically, the nth Maclaurin coefficient of $f(z)$ behaves as stated in (2).

In order to use singularity analysis, we first obtain asymptotic forms of the generating functions introduced above, starting with T. The function T satisfies $f(z, T(z)) = 0$, where

$$f(z, T(z)) = z \sum_{i=0}^{r} \frac{T(z)^i}{i!} - T(z),$$

for all z in a neighborhood of 0.

The minimum of $|z_0|$ among the solutions $(z_0, T(z_0))$ to

$$f(z, T(z)) = f^{(0,1)}(z, T(z)) = 0 \tag{3}$$

gives the dominant singularity of T, which is real and positive.[1] Thus T has the following asymptotic form as $z \to z_0$:

$$T(z) \sim T(z_0) - \left(2\frac{f^{(1,0)}(z_0, T(z_0))}{f^{(0,2)}(z_0, T(z_0))}\right)^{1/2} \sqrt{z_0 - z}.$$

We write this as

$$T(z) \sim y_T - c_T\sqrt{z_0 - z},$$

where

$$y_T = T(z_0) \quad \text{and} \quad c_T = \left(2\frac{f^{(1,0)}(z_0, T(z_0))}{f^{(0,2)}(z_0, T(z_0))}\right)^{1/2}.$$

Now looking at $R(z)$ we have

$$R(z) = z \sum_{i=0}^{r-1} \frac{T(z)^i}{i!} \sim \left(z_0 \sum_{i=0}^{r-1} \frac{y_T^i}{i!}\right) - \left(z_0 \sum_{i=0}^{r-1} \frac{i y_T^{i-1} c_T}{i!}\right) \sqrt{z_0 - z}.$$

Equation (3) implies $z_0 \sum_{i=0}^{r-1} \frac{y_T^i}{i!} = 1$, thus we have

$$R(z) \sim 1 - c_R\sqrt{z_0 - z},$$

where

$$c_R = z_0 \sum_{i=0}^{r-1} \frac{i y_T^{i-1} c_T}{i!}.$$

Applying singularity analysis to this gives, as $n \to \infty$,

$$\frac{r_n}{n!} \sim \frac{c_R z_0^{1/2-n}}{2\sqrt{\pi n^3}}.$$

With the expansion of R we can get the expansion of C:

$$C(z) = \log\frac{1}{1 - R(z)} \sim \frac{1}{2}\log\frac{1}{1 - z/z_0}, \qquad \frac{c_n}{n!} \sim \frac{z_0^{-n}}{2n}.$$

[1] We use the notation $f^{(i,j)}$ to denote the function obtained from f by differentiating i times with respect to the first argument and j times with respect to the second argument.

Finally, using $F(z) = \frac{1}{1-R(z)}$ we have

$$F(z) \sim \frac{1}{c_R\sqrt{z_0}} \frac{1}{(1-z/z_0)^{1/2}}, \qquad \frac{f_n}{n!} \sim \frac{z_0^{-n-1/2}}{c_R\sqrt{\pi n}}.$$

We remark that the numbers y_T, c_T and c_R appear in the statements of some of our main propositions.

3 Single Mapping Parameters

We use the phrase "mapping parameter" to refer to a function $\xi : \mathcal{F}_n^r \to \mathbb{Z}_{\geq 0}$. Several such parameters have been studied previously in [19,22,23] including number of cyclic nodes (nodes on cycles), number of components, and number of nodes with a given number of immediate predecessors. In this section we give a further example of such a parameter: number of components of a given size. We call a component with $\mathfrak{r}$ nodes an $\mathfrak{r}$-*component*.

To obtain basic facts about the distribution associated with a mapping parameter, the following procedure – of which the proof of Proposition 1, below, is an example – may be used; for more background on the technique, see [13]. If ξ is a parameter of interest, we use the generating function $F(z,u)$ for r-mappings with u marking the parameter ξ, that is, $F(z,u) = \sum_{n,k\geq 0} f_{n,k} z^n u^k/n!$, where $f_{n,k}$ is the number of r-mappings on $[1..n]$ with ξ-value k. If Φ is a random element of $\mathcal{F}_n^r$, then we find $\mathbf{E}\left[\xi(\Phi)\right]$ and $\mathbf{P}\left[\xi(\Phi) = a\right]$ directly from $F(z,u)$: Writing $D_u F(z,u)$ for the derivative of $F(z,u)$ with respect to u, we have

$$\mathbf{E}\left[\xi(\Phi)\right] = \frac{[z^n] D_u F(z,u)|_{u=1}}{[z^n]F(z,1)}, \qquad \mathbf{P}\left[\xi(\Phi) = a\right] = \frac{[z^n u^a]F(z,u)}{[z^n]F(z,1)}. \tag{4}$$

Proposition 1. *Let $\xi(\varphi)$ be the number of $\mathfrak{r}$-components in $\varphi \in \mathcal{F}_n^r$. Then, as $n \to \infty$,*

$$\mathbf{E}\left[\xi(\Phi)\right] \sim \frac{c_\mathfrak{r} z_0^\mathfrak{r}}{\mathfrak{r}!}, \quad \textit{and} \quad \mathbf{P}\left[\xi(\Phi) = a\right] \sim \frac{1}{a!}\left(c_\mathfrak{r}\frac{z_0^\mathfrak{r}}{\mathfrak{r}!}\right)^a \exp\left(-c_\mathfrak{r}\frac{z_0^\mathfrak{r}}{\mathfrak{r}!}\right), \quad a \geq 0.$$

Proof. By marking the components of size $\mathfrak{r}$, we have

$$F(z,u) = \exp\left(C(z) + (u-1)c_\mathfrak{r}\frac{z^\mathfrak{r}}{\mathfrak{r}!}\right).$$

Differentiating with respect to the variable u and setting $u = 1$ we obtain

$$D_u F(z,u)|_{u=1} = c_\mathfrak{r}\frac{z^\mathfrak{r}}{\mathfrak{r}!}\frac{1}{1-R(z)} \sim c_\mathfrak{r}\frac{z_0^\mathfrak{r}}{\mathfrak{r}!}F(z), \quad \text{and}$$

$$[u^a]F(z,u) = \frac{1}{a!}\left(c_\mathfrak{r}\frac{z^\mathfrak{r}}{\mathfrak{r}!}\right)^a \exp\left(-c_\mathfrak{r}\frac{z^\mathfrak{r}}{\mathfrak{r}!}\right)\frac{1}{1-R(z)} \sim \frac{1}{a!}\left(c_\mathfrak{r}\frac{z_0^\mathfrak{r}}{\mathfrak{r}!}\right)^a \exp\left(-c_\mathfrak{r}\frac{z_0^\mathfrak{r}}{\mathfrak{r}!}\right)F(z).$$

All that remains is to apply singularity analysis and substitute into (4). □

4 Pairs of Mapping Parameters

In this section we consider, simultaneously, two mapping parameters ξ_1 and ξ_2. (To simplify notation, we use only these two symbols in this section, defining the parameters they represent in each subsection.) Some joint distributions have been studied before, such as the number of components and number of cyclic nodes [15], and size-$\mathfrak{r}$ cycles and components [22]. Here we obtain three kinds of results for new pairs of parameters. Again let Φ be a random element of $\mathcal{F}_n^r$, and let $F(z, u_1, u_2)$ be the generating function for r-mappings, with u_1 marking ξ_1 and u_2 marking ξ_2, that is, $n![z^n u_1^{a_1} u_2^{a_2}]F(z, u_1, u_2)$ is the number of mappings in $\mathcal{F}_n^r$ with ξ_1-value a_1 and ξ_2-value a_2. Then

$$\mathbf{P}\left[\xi_1(\Phi) = a_1 \wedge \xi_2(\Phi) = a_2\right] = \frac{[z^n u_1^{a_1} u_2^{a_2}]\, F(z, u_1, u_2)}{[z^n]F(z, 1, 1)},$$

$$\mathbf{P}\left[\xi_1(\Phi) = a_1 | \xi_2(\Phi) = a_2\right] = \frac{[z^n u_1^{a_1} u_2^{a_2}]\, F(z, u_1, u_2)}{[z^n u_2^{a_2}]F(z, 1, u_2)},$$

$$\mathbf{P}\left[\xi_2(\Phi) = a_2 | \xi_1(\Phi) = a_1\right] = \frac{[z^n u_1^{a_1} u_2^{a_2}]\, F(z, u_1, u_2)}{[z^n u_1^{a_1}]F(z, u_1, 1)},$$

$$\mathbf{E}\left[\xi_1(\Phi) | \xi_2(\Phi) = a_2\right] = \frac{[z^n u_2^{a_2}]\, D_{u_1} F(z, u_1, u_2)|_{u_1=1}}{[z^n u_2^{a_2}]F(z, 1, u_2)}, \quad \text{and}$$

$$\mathbf{E}\left[\xi_2(\Phi) | \xi_1(\Phi) = a_1\right] = \frac{[z^n u_1^{a_1}]\, D_{u_2} F(z, u_1, u_2)|_{u_2=1}}{[z^n u_1^{a_1}]F(z, u_1, 1)}.$$

The denominators have been obtained either in the previous section or in the literature. It remains to obtain asymptotic forms for $[u_1^{a_1} u_2^{a_2}]\, F(z, u_1, u_2)$, $[u_2^{a_2}]\, D_{u_1} F(z, u_1, u_2)|_{u_1=1}$ and $[u_1^{a_1}]\, D_{u_2} F(z, u_1, u_2)|_{u_2=1}$, and apply singularity analysis.

4.1 $\mathfrak{r}$-Components and Components

Proposition 2. *If $\varphi \in \mathcal{F}_n^r$, let $\xi_1(\varphi)$ be the number of $\mathfrak{r}$-components in φ and let $\xi_2(\varphi)$ be the number of components (of any size) in φ. Let*

$$\theta = \left(c_{\mathfrak{r}} \frac{z_0^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1} \binom{a_2}{a_1} \frac{2^{a_1 - a_2}}{a_2!} (a_2 - a_1).$$

Then, as $n \to \infty$,

$$\mathbf{P}\left[\xi_1(\Phi) = a_1 \wedge \xi_2(\Phi) = a_2\right] \sim \theta c_R \sqrt{\pi z_0}\, n^{-1/2} \log^{a_2 - a_1 - 1} n,$$

$$\mathbf{P}\left[\xi_1(\Phi) = a_1 | \xi_2(\Phi) = a_2\right] \sim \theta 2^{a_2} (a_2 - 1)! \log^{-a_1} n,$$

$$\mathbf{P}\left[\xi_2(\Phi) = a_2 | \xi_1(\Phi) = a_1\right] \sim \theta \frac{c_R \sqrt{\pi z_0}\, a_1! \exp\left(c_{\mathfrak{r}} \frac{z_0^{\mathfrak{r}}}{\mathfrak{r}!}\right)}{\left(c_{\mathfrak{r}} \frac{z_0^{\mathfrak{r}}}{\mathfrak{r}}!\right)^{a_1}} n^{-1/2} \log^{a_2 - a_1 - 1} n,$$

$$\mathbf{E}\left[\xi_1(\Phi) | \xi_2(\Phi) = a_2\right] \sim \frac{2 c_{\mathfrak{r}} z_0^{\mathfrak{r}}}{\mathfrak{r}!} (a_2 - 1) \frac{1}{\log n}, \quad \textit{and}$$

$$\mathbf{E}\left[\xi_2(\Phi) | \xi_1(\Phi) = a_1\right] \sim \frac{1}{2} \log n, \qquad 0 \le a_1 < a_2.$$

Proof. Let u_1 mark the number of $\mathfrak{r}$-components, and let u_2 mark the number of components. We have

$$F(z, u_1, u_2) = \exp\left(u_2\left(C(z) + (u_1 - 1)c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)\right).$$

First we have

$$\begin{aligned}[u_1^{a_1}u_2^{a_2}]F(z, u_1, u_2) &= [u_1^{a_1}]\frac{1}{a_2!}\left(C(z) + (u_1 - 1)c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_2} \\ &= \frac{1}{a_2!}\left(c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1}\left(C(z) - c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_2-a_1}\binom{a_2}{a_1} \\ &\sim \left(c_{\mathfrak{r}}\frac{z_0^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1}\binom{a_2}{a_1}\frac{2^{a_1-a_2}}{a_2!}\left(\log\frac{1}{1 - z/z_0}\right)^{a_2-a_1}.\end{aligned}$$

Then we have

$$\begin{aligned}[u_2^{a_2}]D_{u_1}F(z, u_1, u_2)|_{u_1=1} &= [u_2^{a_2}]\exp\left(u_2C(z)\right)u_2c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!} = c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\frac{C(z)^{a_2-1}}{(a_2-1)!} \\ &\sim c_{\mathfrak{r}}\frac{z_0^{\mathfrak{r}}}{\mathfrak{r}!}\frac{2^{1-a_2}}{(a_2-1)!}\left(\log\frac{1}{1 - z/z_0}\right)^{a_2-1}.\end{aligned}$$

To extract coefficients from the previous two generating functions we can use the fact from [25] that $[z^n]\log^{\beta}(1-z)^{-1} = \frac{\beta!}{n!}\begin{bmatrix} n \\ \beta \end{bmatrix} \sim \beta n^{-1}\log^{\beta-1} n$, where $\begin{bmatrix} n \\ \beta \end{bmatrix}$ is the number of permutations of n elements with β disjoint cycles (Stirling numbers of the first kind). Lastly,

$$\begin{aligned}&[u_1^{a_1}]D_{u_2}F(z, u_1, u_2)|_{u_2=1} \\ =&[u_1^{a_1}]\exp\left(C(z) + (u_1 - 1)c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)\left(C(z) + (u_1 - 1)c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right) \\ =&\exp\left(C(z) - c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)\left(\frac{1}{a_1!}\left(C(z) - c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)\left(c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1} + \frac{1}{(a_1-1)!}c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\left(c_{\mathfrak{r}}\frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1-1}\right) \\ \sim&\frac{1}{2}\exp\left(-c_{\mathfrak{r}}\frac{z_0^{\mathfrak{r}}}{\mathfrak{r}!}\right)\left(c_{\mathfrak{r}}\frac{z_0^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1}\frac{1}{a_1!}\frac{1}{c_R\sqrt{z_0}\sqrt{1 - z/z_0}}\log\frac{1}{1 - z/z_0}.\end{aligned}$$

□

4.2 $\mathfrak{r}$-Components and Cyclic Nodes

Proposition 3. *If $\varphi \in \mathcal{F}_n^r$, let $\xi_1(\varphi)$ be the number of $\mathfrak{r}$-components in φ and let $\xi_2(\varphi)$ be the number of cyclic nodes in φ. Let*

$$\theta = \sum_{j=0}^{a_2} j\left(\sum_{k\geq 0}\frac{(-z_0^{\mathfrak{r}})^k}{k!}\left[u_2^{a_2-j}\right]\left([z^{\mathfrak{r}}]\log\frac{1}{1 - u_2R(z)}\right)^{k+a_1}\right).$$

Then, as $n \to \infty$,

$$\begin{aligned}
\mathbf{P}\left[\xi_1(\Phi) = a_1 \wedge \xi_2(\Phi) = a_2\right] &\sim \theta \frac{c_R^2 z_0^{\mathfrak{r} a_1 + 1}}{2a_1!} n^{-1},\\
\mathbf{P}\left[\xi_1(\Phi) = a_1 | \xi_2(\Phi) = a_2\right] &\sim \theta \frac{z_0^{\mathfrak{r} a_1}}{a_2 a_1!},\\
\mathbf{P}\left[\xi_2(\Phi) = a_2 | \xi_1(\Phi) = a_1\right] &\sim \theta \frac{c_R^2 z_0}{2} \left(\frac{\mathfrak{r}!}{c_\mathfrak{r}}\right)^{a_1} \exp\left(c_\mathfrak{r} \frac{z_0^\mathfrak{r}}{\mathfrak{r}!}\right) n^{-1},\\
\mathbf{E}\left[\xi_1(\Phi) | \xi_2(\Phi) = a_2\right] &\sim \sum_{j=0}^{a_2-1} \frac{j z_0^\mathfrak{r}}{a_2(a_2-j)} [z^\mathfrak{r}] R(z)^{a_2-j}, \quad \textit{and}\\
\mathbf{E}\left[\xi_2(\Phi) | \xi_1(\Phi) = a_1\right] &\sim \frac{\sqrt{\pi}}{c_R \sqrt{z_0}} n^{1/2}, \qquad a_1 \geq 0,\ a_2 \geq 1.
\end{aligned}$$

Proof. Let u_1 mark the number of $\mathfrak{r}$-components and let u_2 mark the number of cyclic nodes. We have

$$F(z, u_1, u_2) = \exp\left(\log\left(\frac{1}{1 - u_2 R(z)}\right) + (u_1 - 1) z^\mathfrak{r} \left([z^\mathfrak{r}] \log \frac{1}{1 - u_2 R(z)}\right)\right).$$

From this, we first get

$$\begin{aligned}
[u_2^{a_2} u_1^{a_1}] F(z, u_1, u_2) &= [u_2^{a_2}] \frac{1}{1 - u_2 R(z)} \exp\left(-z^\mathfrak{r} [z^\mathfrak{r}] \log \frac{1}{1 - u_2 R(z)}\right) \frac{1}{a_1!} \left(z^\mathfrak{r} [z^\mathfrak{r}] \log \frac{1}{1 - u_2 R(z)}\right)^{a_1}\\
&= \frac{z^{\mathfrak{r} a_1}}{a_1!} \sum_{j=0}^{a_2} R(z)^j \sum_{k \geq 0} \frac{(-z^\mathfrak{r})^k}{k!} [u_2^{a_2-j}] \left([z^\mathfrak{r}] \log \frac{1}{1 - u_2 R(z)}\right)^{k + a_1}\\
&\sim \frac{z_0^{\mathfrak{r} a_1}}{a_1!} \sum_{j=0}^{a_2} \left(\sum_{k \geq 0} \frac{(-z_0^\mathfrak{r})^k}{k!} [u_2^{a_2-j}] \left([z^\mathfrak{r}] \log \frac{1}{1 - u_2 R(z)}\right)^{k + a_1}\right) \left(1 - j c_R \sqrt{z_0 - z}\right).
\end{aligned}$$

Now

$$\begin{aligned}
[u_2^{a_2}] D_{u_1} F(z, u_1, u_2)|_{u_1 = 1} &= [u_2^{a_2}] \frac{1}{1 - u_2 R(z)} z^\mathfrak{r} \left([z^\mathfrak{r}] \log \frac{1}{1 - u_2 R(z)}\right)\\
&= \sum_{j=0}^{a_2-1} R(z)^j z^\mathfrak{r} [z^\mathfrak{r}] R(z)^{a_2-j} / (a_2 - j)\\
&\sim \sum_{j=0}^{a_2-1} \left(1 - j c_R \sqrt{z_0 - z}\right) z_0^\mathfrak{r} [z^\mathfrak{r}] R(z)^{a_2-j} / (a_2 - j),
\end{aligned}$$

and

$$
\begin{aligned}
&[u_1^{a_1}] D_{u_2} F(z,u_1,u_2)|_{u_2=1} \\
=&[u_1^{a_1}] \left(\frac{R(z)}{(1-R(z))^2} \exp\left((u_1-1)c_{\mathfrak{r}} \frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right) + \frac{1}{1-R(z)} \exp\left((u_1-1)c_{\mathfrak{r}} \frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right) (u_1-1) z^{\mathfrak{r}} [z^{\mathfrak{r}}] \frac{R(z)}{1-R(z)} \right) \\
=&\frac{R(z)}{(1-R(z))^2} \exp\left(-c_{\mathfrak{r}} \frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right) \frac{1}{a_1!} \left(c_{\mathfrak{r}} \frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1} + \frac{z^{\mathfrak{r}}}{1-R(z)} \left([z^{\mathfrak{r}}] \frac{R(z)}{1-R(z)}\right) \exp\left(-c_{\mathfrak{r}} \frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right) \\
&\qquad \left(\frac{1}{(a_1-1)!} \left(c_{\mathfrak{r}} \frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1-1} - \frac{1}{a_1!} \left(c_{\mathfrak{r}} \frac{z^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1} \right) \\
\sim& \exp\left(-c_{\mathfrak{r}} \frac{z_0^{\mathfrak{r}}}{\mathfrak{r}!}\right) \left(c_{\mathfrak{r}} \frac{z_0^{\mathfrak{r}}}{\mathfrak{r}!}\right)^{a_1} \frac{1}{a_1!} \frac{1}{c_R^2 z_0 (1-z/z_0)}.
\end{aligned}
$$

□

4.3 $\mathfrak{r}_1$-Cycles and $\mathfrak{r}_2$-Cycle Trees

Proposition 4. *If $\varphi \in \mathcal{F}_n^r$, let $\xi_1(\varphi)$ be the number of $\mathfrak{r}_1$-cycles (cycles with $\mathfrak{r}_1$ cyclic nodes) in φ and let $\xi_2(\varphi)$ be the number of $\mathfrak{r}_2$-cycle trees ($\mathfrak{r}$-node trees rooted on a cycle) in φ. Let $p_0 = r_{\mathfrak{r}_2} \frac{z_0^{\mathfrak{r}_2}}{\mathfrak{r}_2!}$. Then, as $n \to \infty$,*

$$\mathbf{E}\left[\xi_1(\Phi) | \xi_2(\Phi) = a_2\right] \sim \frac{p_0^{a_2+2}}{\mathfrak{r}_1 (a_2+1)} \sum_{j=0}^{\mathfrak{r}_1} \sum_{i=0}^{\mathfrak{r}_1 - j} i p_0^{\mathfrak{r}_1 - i - 1} + (a_2 - j + 1) p_0^{\mathfrak{r}_1 - i - 2},$$

$$\mathbf{E}\left[\xi_2(\Phi) | \xi_1(\Phi) = a_1\right] \sim \frac{p_0 \sqrt{\pi}}{c_R z_0^{1/2}} n^{1/2}, \qquad a_1 \geq 0,\ a_2 \geq r.$$

Proof. Let u_1 mark the number of $\mathfrak{r}_1$-cycles and let u_2 mark the number of $\mathfrak{r}_2$-cycle trees. We have

$$F(z,u_1,u_2) = \exp\left(\log \frac{1}{1-(R(z)+(u_2-1)p)} + (u_1-1) \frac{(R(z)+(u_2-1)p)^{\mathfrak{r}_1}}{\mathfrak{r}_1} \right).$$

This gives

$$
\begin{aligned}
&[u_2^{a_2}] D_{u_1} F(z,u_1,u_2)|_{u_1=1} \\
=&[u_2^{a_2}] \frac{1}{1-(R(z)+(u_2-1)p)} \frac{(R(z)+(u_2-1)p)^{\mathfrak{r}_1}}{\mathfrak{r}_1} = \frac{p^{a_2}}{\mathfrak{r}_1} \sum_{j=0}^{\mathfrak{r}_1} (R(z)-p)^{\mathfrak{r}_1 - j} \frac{1}{(1-R(z)+p)^{a_2 - j + 1}} \\
\sim& \frac{1}{\mathfrak{r}_1} \left(\left(\sum_{j=0}^{\mathfrak{r}_1} \sum_{i=0}^{\mathfrak{r}_1 - j} p_0^{\mathfrak{r}_1 - i - 1} \right) - \left(c_R \sum_{j=0}^{\mathfrak{r}_1} \sum_{i=0}^{\mathfrak{r}_1 - j} i p_0^{\mathfrak{r}_1 - i - 1} + (a_2 - j + 1) p_0^{\mathfrak{r}_1 - i - 2} \right) \sqrt{z_0 - z} \right),
\end{aligned}
$$

and

$$
\begin{aligned}
&[u_1^{a_1}] D_{u_2} F(z,u_1,u_2)|_{u_2=1} \\
=&[u_1^{a_1}] p F(z,u_1,1) \left(\frac{1}{1-R(z)} + (u_1-1) R(z)^{\mathfrak{r}_1 - 1} \right) \\
=&p \left(\frac{1}{1-R(z)} - R(z)^{\mathfrak{r}_1 - 1} \right) [u_1^{a_1}] F(z,u_1,1) + p R(z)^{\mathfrak{r}_1 - 1} [u_1^{a_1 - 1}] F(z,u_1,1) \\
\sim& \frac{\mathfrak{r}_1^{-a_1}}{a_1!} \exp\left(-\frac{1}{\mathfrak{r}_1}\right) \frac{p_0}{c_R^2 z_0 (1-z/z_0)}.
\end{aligned}
$$

□

Remark 1. We note that $\xi_1(\Phi)$ and $\xi_2(\Phi)$ have not been asymptotically independent for any of the pairs of ξ_1, ξ_2 for which the joint probability was found in this section.

5 Extremal Parameters

In this section we obtain results for expected maximum–type properties of r-mappings. All parameters in this section were studied in the general mapping setting in [12]. The expected size of the longest cycle in an r-mapping was found in [19] to be the Golomb–Dickman constant (OEIS A084945) times the expected number of cyclic nodes.

Again, let Φ be a random element of $\mathcal{F}_n^r$.

5.1 Largest Tree

Let $\xi(\varphi)$ be the size of the largest tree rooted on a cycle in $\varphi \in \mathcal{F}_n^r$. Then Theorem 3 in [26] applies, and we have, as $n \to \infty$,

$$\mathbf{E}[\xi(\Phi)] \sim c_1 n, \qquad \mathrm{Var}[\xi(\Phi)] \sim c_2 n^2,$$

where, with $\alpha = 1/2, \gamma = -1$,

$$c_1 = \frac{1}{\alpha\gamma}\int_0^\infty \left(\left(1 - \frac{1}{\Gamma(-\alpha)}\int_x^\infty \frac{e^{-t}}{t^{\alpha+1}}dt\right)^\gamma - 1\right) dx \doteq 0.4834983472,$$

and

$$c_2 = \frac{2}{\alpha\gamma(1-\alpha\gamma)}\int_0^\infty \left(\left(1 - \frac{1}{\Gamma(-\alpha)}\int_x^\infty \frac{e^{-t}}{t^{\alpha+1}}dt\right)^\gamma - 1\right) x dx - c_1^2 \doteq 0.04946985228.$$

These are the same results as for general mappings. Results on the limiting distribution are also available in [26].

5.2 Largest Component

We adapt the proof in [12] for unrestricted mappings to the case of r-mappings. We use the following notation: If $A(z) = a_0 + a_1 z + a_2 z^2 + \cdots$ is a power series,

$$\mathbf{T}_m[A(z)] = \sum_{n \le m} a_n z^n, \qquad \mathbf{R}_m[A(z)] = \sum_{n > m} a_n z^n.$$

Let $\xi(\varphi)$ be the size of the largest component in $\varphi \in \mathcal{F}_n^r$, and let $\Xi(z) = \sum_{n\ge 0} \xi_n \frac{z^n}{n!}$ where $\xi_n = \sum_{\varphi\in\mathcal{F}_n^r} \xi(\varphi)$.

Proposition 5. *As $n \to \infty$,*

$$\mathbf{E}[\xi(\Phi)] \sim cn,$$

where

$$c = -2\log(z_0)\int_0^\infty 1 - \exp\left(-\int_x^\infty \frac{z_0^u}{2u}du\right) dx.$$

Proof. Since $F(z) = \exp(C(z)) = (1 - R(z))^{-1}$, we have

$$\Xi(z) = \sum_{m \geq 0} \left(F(z) - \exp\left(\mathbf{T}_m[C(z)]\right)\right) = \frac{1}{1 - R(z)} \sum_{m \geq 0} \left(1 - \exp\left(\mathbf{T}_m[C(z)] - C(z)\right)\right).$$

Set $z = z_0^{1+y}$. As discussed in [12], we assume that we only need to analyze $\Xi(z)$ on the real line. Thus y is real and

$$\mathbf{T}_m[C(z)] - C(z) \approx -\sum_{n>m} \frac{z_0^{yn}}{2n} \approx -\int_m^\infty \frac{z_0^{yv}}{2v} dv = -\int_{my}^\infty \frac{z_0^u}{2u} du.$$

Now we substitute back and use Euler-Maclaurin summation:

$$\begin{aligned}
\Xi(z) &\sim \frac{1}{1 - R(z)} \sum_{m \geq 0} \left(1 - \exp\left(-\int_{my}^\infty \frac{z_0^u}{2u} du\right)\right) \\
&\sim \frac{1}{1 - R(z)} \int_0^\infty 1 - \exp\left(-\int_{wy}^\infty \frac{z_0^u}{2u} du\right) dw \\
&= \frac{1}{1 - R(z)} \int_0^\infty 1 - \exp\left(-\int_x^\infty \frac{z_0^u}{2u} du\right) \frac{dx}{y} \\
&= \frac{1}{1 - R(z)} \frac{z_0 \log(z_0)}{z - z_0} \int_0^\infty 1 - \exp\left(-\int_x^\infty \frac{z_0^u}{2u} du\right) dx, \\
&\sim \frac{-\log(z_0)}{c_R \sqrt{z_0}} \frac{1}{(1 - z/z_0)^{3/2}} \int_0^\infty 1 - \exp\left(-\int_x^\infty \frac{z_0^u}{2u} du\right) dx.
\end{aligned}$$

We now have a form for $\Xi(z)$ to which singularity analysis can be directly applied. □

6 Tail Length and Cycle Length

A node parameter is a function $\xi : \{\psi \in \varphi : \varphi \in \mathcal{F}_n^r\} \to \mathbb{Z}_{\geq 0}$, where $\psi \in \varphi$ means ψ is a node in (the functional graph of) φ. This section focuses on the two node parameters tail length ξ_1 (the distance from a node to the root of its tree) and cycle length ξ_2 (the size of the cycle in the component containing a node). We explain at the end of this section how the distribution of these two parameters gives information on the running time of Brent's variant of Pollard's rho algorithm.

Let Ψ be a random node in a mapping in $\mathcal{F}_n^r$, where all n^{n+1} such nodes are equally likely. The limiting joint distribution of $\xi_1(\Psi)$ and $\xi_2(\Psi)$ has been obtained in [21]. This section contains rederivations using relatively elementary methods without leaving the framework of [12].

Proposition 6. *If $a_2 \geq 1$, as $n \to \infty$,*

$$\mathbf{E}\left[\xi_1(\Psi)\right] \sim \left(\frac{\sqrt{\pi}}{\sqrt{z_0}\, c_R^3} \sum_{i=0}^{r-2} \frac{y_T^{i+1}}{i!}\right) n^{1/2},$$

$$\mathbf{P}\left[\xi_1(\Psi) = a_1\right] \sim \begin{cases} \left(\frac{\sqrt{\pi z_0}}{c_R} \sum_{i=0}^{r-2} \frac{y_T^{i+1}}{i!}\right) n^{-1/2} & \text{if } a_1 \geq 1, \\ \frac{\sqrt{\pi}}{c_R \sqrt{z_0}} n^{-1/2} & \text{if } a_1 = 0, \end{cases}$$

$$\mathbf{E}\left[\xi_2(\Psi)\right] \sim \frac{\sqrt{\pi}}{2 c_R \sqrt{z_0}} n^{1/2},$$

$$\mathbf{P}\left[\xi_2(\Psi) = a_2\right] \sim \frac{c_R \sqrt{\pi z_0}}{2} n^{-1/2},$$

$$\mathbf{P}\left[\xi_1(\Psi) = a_1 \wedge \xi_2(\Psi) = a_2\right] \sim \begin{cases} z_0 \sum_{i=0}^{r-2} \frac{y_T^{i+1}}{i!} n^{-1} & \text{if } a_1 \geq 1, \\ n^{-1} & \text{if } a_1 = 0, \end{cases}$$

$$\mathbf{P}\left[\xi_1(\Psi) = a_1 | \xi_2(\Psi) = a_2\right] \sim \begin{cases} \left(\frac{2\sqrt{z_0}}{c_R \sqrt{\pi}} \sum_{i=0}^{r-2} \frac{y_T^{i+1}}{i!}\right) n^{-1/2} & \text{if } a_1 \geq 1, \\ \frac{2}{c_R \sqrt{\pi z_0}} n^{-1/2} & \text{if } a_1 = 0, \end{cases}$$

$$\mathbf{P}\left[\xi_2(\Psi) = a_2 | \xi_1(\Psi) = a_1\right] \sim \frac{c_R \sqrt{z_0}}{\sqrt{\pi}} n^{-1/2},$$

$$\mathbf{E}\left[\xi_1(\Psi) | \xi_2(\Psi) = a_2\right] \sim \left(\frac{4}{c_R^3 \sqrt{\pi z_0}} \sum_{i=0}^{r-2} \frac{y_T^{i+1}}{i!}\right) n^{1/2}, \quad \text{and}$$

$$\mathbf{E}\left[\xi_2(\Psi) | \xi_1(\Psi) = a_1\right] \sim \frac{2}{c_R \sqrt{\pi z_0}} n^{1/2}.$$

Proof. Let

$$\Xi_F(z, u_1, u_2) = \sum_{n \geq 0} \sum_{a_1, a_2 \geq 0} \xi_F(n, a_1, a_2) u_1^{a_1} u_2^{a_2} \frac{z^n}{n!},$$

where $\xi_F(n, a_1, a_2)$ is the number of nodes ψ in mappings in $\mathcal{F}_n^r$ such that $\xi_1(\psi) = a_1$ and $\xi_2(\psi) = a_2$, and let

$$\Xi_C(z, u_1, u_2) = \sum_{n \geq 0} \sum_{a_1, a_2 \geq 0} \xi_C(n, a_1, a_2) u_1^{a_1} u_2^{a_2} \frac{z^n}{n!},$$

where $\xi_C(n, a_1, a_2)$ is the number of nodes ψ in size-n components such that $\xi_1(\psi) = a_1$ and $\xi_2(\psi) = a_2$. Define $\Xi_T(z, u_1)$ and $\Xi_R(z, u_1)$ analogously with only u_1 marking tail length, since cycle length is defined for nodes in a component.

As proof of the proposition, we merely obtain an explicit expression for $\Xi_F(z, u_1, u_2)$; respective extraction of coefficients provides the stated results.

First we look at $\Xi_T(z, u_1)$. A node in an r-ary tree is either a root with tail length 0, or a node in a subtree of the root, with tail length equal to the tail

length in that subtree, plus 1. This yields

$$\Xi_T(z,u_1) = T(z) + zu_1\Xi_T(z,u_1)\sum_{i=0}^{r-1}\frac{T(z)^i}{i!} = \frac{T(z)}{1-u_1R(z)}.$$

Similarly

$$\Xi_R(z,u_1) = R(z) + zu_1\Xi_T(z,u_1)\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}.$$

Now we wish to use $\Xi_R(z,u)$ to obtain $\Xi_C(z,u)$. A labeled tree will appear once in a component for every combination of trees that make up the rest of the cycle, and the nodes in a tree have cycle length a_2 if the tree is part of a cycle with a_2-1 other trees, so

$$\Xi_C(z,u_1,u_2) = \Xi_R(z,u_1)\frac{u_2}{1-u_2R(z)}.$$

Now we note that $n![z^n]\Xi_F(z,u_1,u_2)$ is a sum of $\xi_C(i,a_1,a_2)u_1^{a_1}u_2^{a_2}$ weighted by the number of times a component of size i can appear in a mapping in $\mathcal{F}_n^r$:

$$n![z^n]\Xi_F(z,u_1,u_2) = \sum_{a_1,a_2\geq 0}\sum_{i=0}^{n}\binom{n}{i}\xi_C(i,a_1,a_2)u_1^{a_1}u_2^{a_2}f_{n-i}.$$

This implies

$$\begin{aligned}\Xi_F(z,u_1,u_2) &= F(z)\Xi_C(z,u_1,u_2)\\ &= \frac{1}{1-R(z)}\left(R(z)+\frac{zu_1T(z)}{1-u_1R(z)}\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}\right)\frac{u_2}{1-u_2R(z)}.\end{aligned}$$

We note that because of the form of this generating function, the asymptotic results stated in the proposition are largely free of a_1 and a_2. □

If (but not iff) Pollard's rho algorithm [18] on a composite integer m terminates, a cycle has been found, via Brent's algorithm [27], in the functional graph of the mapping $x \mapsto x^2 - 1 \bmod k$, for some k where $k|m$. If k is prime, the images of this mapping have at most $r = 2$ preimages. If $x \in \mathbb{Z}_k$ is the starting value for the sequence of iterates, then Brent's algorithm on the mapping will find a cycle after $\xi_2(x) + 2^{\lceil \lg(\max(\xi_1(x),\xi_2(x)))\rceil}$ iterations. We note that we have the simple bounds

$$\xi_2(x)+\max(\xi_1(x),\xi_2(x)) \leq \xi_2(x)+2^{\lceil \lg(\max(\xi_1(x),\xi_2(x)))\rceil} \leq \xi_2(x)+2\max(\xi_1(x),\xi_2(x)).$$

In the following proposition we obtain $\mathbf{E}\left[\max\left(\xi_1\left(\Psi\right),\xi_2\left(\Psi\right)\right)\right]$ in terms of the generating functions defined up to now, and then we use it in Corollary 1 to give bounds on the expected number of iterations of Brent's algorithm on 2-mappings.

Proposition 7. *As $n \to \infty$,*

$$\mathbf{E}\left[\max(\xi_1(\Psi), \xi_2(\Psi))\right] \sim \left(\frac{3\sqrt{\pi}}{2c_R^3\sqrt{z_0}} \sum_{i=0}^{r-2} \frac{y_T^{i+1}}{i!}\right) n^{1/2}.$$

Proof. Let $\Xi_F(z, u_1, u_2)$ be defined as in the proof of Proposition 6 above, and let

$$M(z) = \sum_{n\geq 0} \sum_{a_1,a_2\geq 0} \max(a_1, a_2)\xi_F(n, a_1, a_2)\frac{z^n}{n!}.$$

Rearranging the double sum we get

$$M(z) = \sum_{n\geq 0} \left(2 \sum_{a_1>a_2>0} a_1\xi_F(n, a_1, a_2) + \sum_{a_2>0} a_2\xi_F(n, 0, a_2) + \sum_{a_1>0} a_1\xi_F(n, a_1, 0) + \sum_{j\geq 0} j\xi_F(n, j, j)\right) \frac{z^n}{n!}.$$

The fact that $\sum_{a_1>a_2>0} a_1\xi_F(n, a_1, a_2) = \sum_{a_2>a_1>0} a_2\xi_F(n, a_1, a_2)$ follows from the expression obtained for $\Xi_F(z, u_1, u_2)$ in Proposition 6. Now,

$$\sum_{n\geq 0}\sum_{a_2>0} a_2\xi_F(n, 0, a_2)\frac{z^n}{n!} = \sum_{n\geq 0}\sum_{a_2\geq 0} a_2\xi_F(n, 0, a_2)\frac{z^n}{n!} = D_{u_2}\Xi_F(z, 0, u_2)\Big|_{u_2=1}$$
$$= D_{u_2}\frac{R(z)}{1-R(z)}\frac{u_2}{1-u_2R(z)}\Big|_{u_2=1} = \frac{R(z)}{(1-R(z))^3}.$$

To obtain $\sum_{j\geq 0} j\xi_F(n, j, j)$, we extract the diagonal from $u_1 D_{u_1}\Xi_F(z, u_1, u_2)$ using a standard trick:

$$\sum_{n\geq 0}\sum_{j\geq 0} j\xi_F(n, j, j)\frac{z^n}{n!} = [u_1^0]\left((u_1 D_{u_1}\Xi_F(z, u_1, u_2))\,|_{u_2=u_1^{-1}}\right)$$
$$= \left(\frac{zT(z)}{1-R(z)}\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}\right)[u_1^0]\left(\sum_{j\geq 0}(j+1)u_1^j R(z)^j\right)\left(\sum_{j\geq 0} u_1^{-j}R(z)^j\right)$$
$$= \left(\frac{zT(z)}{1-R(z)}\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}\right)\left(\sum_{j\geq 0}(j+1)R(z)^{2j}\right)$$
$$= \left(\frac{zT(z)}{1-R(z)}\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}\right)\frac{1}{(1-R(z)^2)^2},$$

where $[z^0]f(z)$ is the constant term of the formal Laurent series $f(z)$. Finally, we use a similar technique to obtain $\sum_{a_1>a_2>0} a_1\xi_F(n, a_1, a_2)$. As in [28], if $f(z) = \sum_{n=-\infty}^{\infty} f_n z^n$ is a formal Laurent series, we use the notation $\left[\frac{z}{1-z}\right] f(z)$

to denote $f_1 + f_2 + \cdots$. We have

$$\begin{aligned}
&\sum_{n\geq 0}\sum_{a_1>a_2>0} a_1\xi_F(n,a_1,a_2)\frac{z^n}{n!} = \left[\frac{u_1}{1-u_1}\right]\left((u_1D_{u_1})\Xi_F(z,u_1,u_2)|_{u_2=u_1^{-1}}\right)\\
&= \left(\frac{zT(z)}{1-R(z)}\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}\right)\left[\frac{u_1}{1-u_1}\right]\left(\sum_{j\geq 0}(j+1)u_1^jR(z)^j\right)\left(\sum_{j\geq 0}u_1^{-j}R(z)^j\right)\\
&= \left(\frac{zT(z)}{1-R(z)}\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}\right)\sum_{n\geq 0}R(z)^n\sum_{j=\lfloor n/2\rfloor+1}^{n}(j+1)\\
&= \left(\frac{zT(z)}{1-R(z)}\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}\right)\left(\frac{5+10R(z)-3R(z)^2}{16(1-R(z))^3}+\frac{-5-3R(z)}{16(1+R(z))^2}\right).
\end{aligned}$$

We note that $\sum_{a_1>0} a_1\xi_F(n,a_1,0)$ is just 0. Putting all this together, we have

$$\begin{aligned}
M(z) &= \frac{R(z)}{(1-R(z))^3} + \left(\frac{zT(z)}{1-R(z)}\sum_{i=0}^{r-2}\frac{T(z)^i}{i!}\right)\left(\frac{1}{(1-R(z)^2)^2}+2\frac{5+10R(z)-3R(z)^2}{16(1-R(z))^3}\right.\\
&\qquad\qquad \left. + 2\frac{-5-3R(z)}{16(1+R(z))^2}\right)\\
&\sim z_0y_T\sum_{i=0}^{r-2}\frac{y_T^i}{i!}\frac{3/2}{(1-R(z))^4}.
\end{aligned}$$

The result now follows since

$$\mathbf{E}\left[\max(\xi_1(\Psi),\xi_2(\Psi))\right] = \frac{[z^n]M(z)}{n[z^n]F(z)}. \qquad \square$$

This allows us to state the following corollary.

Corollary 1. *Let $g(n)$ be the expected number of iterations Brent's algorithm requires to find a cycle in a random mapping in $\mathcal{F}_n^2$ starting from a uniform random point. Then there exist functions g_L and g_U such that, for all n,*

$$g_L(n) \leq g(n) \leq g_U(n),$$

and, as $n \to \infty$,

$$g_L(n) \sim \frac{5}{4}\frac{\sqrt{\left(1+\sqrt{2}\right)\pi}}{2^{3/4}}n^{1/2} \qquad g_U(n) \sim 2^{1/4}\sqrt{\left(1+\sqrt{2}\right)\pi}\,n^{1/2}.$$

Proof. This follows from the preceding discussion and the following fact, from [19]:

$$\mathbf{E}\left[\xi_2(\Psi)\right] \sim \frac{\sqrt{\pi}}{2c_R\sqrt{z_0}}n^{1/2},$$

as $n \to \infty$. $\square$

7 Conclusion

We have shown in this paper how the generating function framework described in [12] and [13] can be used to solve problems related to r-mappings, some motivated by cryptographic applications, in a relatively unified manner. Generating functions can often lead to the simplest proofs because of their ease of manipulation and the existence of asymptotic moment extraction theorems, and sometimes they provide the only known solution to problems in asymptotic counting and probability. We hope that this paper may encourage more work towards understanding random mappings.

A natural continuation of this work would be to find limiting distributions for all parameters mentioned, since limiting distributions have already been obtained in the literature for a number of parameters and pairs of parameters. In addition, there are two parameters from [12], largest tail length and largest rho length, which could be studied in the r-mapping setting.

Acknowledgements. We would like to thank Igor Shparlinski for suggesting this line of research and its applications to cryptography, and anonymous referees for their helpful comments.

References

1. Aldous, D.J.: Exchangeability and related topics. In: École D'été de Probabilités de Saint-Flour, XIII—1983. Lecture Notes in Math., vol. 1117, pp. 1–198. Springer, Berlin (1985)
2. Arratia, R., Tavaré, S.: Limit theorems for combinatorial structures via discrete process approximations. Random Structures Algorithms 3(3), 321–345 (1992)
3. Baron, G., Drmota, M., Mutafchiev, L.: Predecessors in random mappings. Combin. Probab. Comput. 5, 317–335 (1995)
4. DeLaurentis, J.M.: Components and Cycles of a Random Function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 231–242. Springer, Heidelberg (1988)
5. Drmota, M., Gittenberger, B.: Strata of random mappings—a combinatorial approach. Stochastic Process. Appl. 82, 157–171 (1999)
6. Hansen, J.C.: A functional central limit theorem for random mappings. Ann. Probab. 17, 317–332 (1989)
7. Harris, B.: Probability distributions related to random mappings. The Annals of Mathematical Statistics 31, 1045–1062 (1960)
8. Kolchin, V.F.: A problem of the allocation of particles in cells and random mappings. Theory Probab. Appl. 21, 48–63 (1976)
9. Kolchin, V.F.: Random mappings. Translation Series in Mathematics and Engineering. Optimization Software Inc. Publications Division, New York (1986); Translated from the Russian, with a foreword by Varadhan, S.R.S.
10. Mutafchiev, L.R.: Limit properties of components of random mappings. C. R. Acad. Bulgare Sci. 31(10), 1257–1260 (1978)
11. Mutafchiev, L.R.: The limit distribution of the number of nodes in low strata of a random mapping. Statist. Probab. Lett. 7(3), 247–251 (1988)

12. Flajolet, P., Odlyzko, A.M.: Random Mapping Statistics. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 329–354. Springer, Heidelberg (1990)
13. Flajolet, P., Sedgewick, R.: Analytic combinatorics. Cambridge University Press, Cambridge (2009)
14. Aldous, D., Pitman, J.: The asymptotic distribution of the diameter of a random mapping. C. R. Math. Acad. Sci. Paris 334(11), 1021–1024 (2002)
15. Drmota, M., Soria, M.: Marking in combinatorial constructions: generating functions and limiting distributions. Theoret. Comput. Sci. 144(1-2), 67–99 (1995)
16. Hansen, J.C., Jaworski, J.: Local properties of random mappings with exchangeable in-degrees. Adv. in Appl. Probab. 40(1), 183–205 (2008)
17. Hansen, J.C., Jaworski, J.: Random mappings with a given number of cyclical points. Ars Combin. 94, 341–359 (2010)
18. Pollard, J.M.: A Monte Carlo method for factorization. Nordisk Tidskr. Informationsbehandling (BIT) 15(3), 331–334 (1975)
19. Arney, J., Bender, E.A.: Random mappings with constraints on coalescence and number of origins. Pacific J. Math. 103(2), 269–294 (1982)
20. Gittenberger, B.: On the number of predecessors in constrained random mappings. Statist. Probab. Lett. 36(1), 29–34 (1997)
21. Rubin, H., Sitgreaves, R.: Probability distributions related to random transformations on a finite set. Technical Report 19a, Applied Mathematics and Statistics Laboratory, Stanford (1954)
22. Stepanov, V.E.: Limit distributions of certain characteristics of random mappings. Teor. Verojatnost. i Primenen. 14, 639–653 (1969)
23. Drmota, M., Soria, M.: Images and preimages in random mappings. SIAM J. Discrete Math. 10(2), 246–269 (1997)
24. Flajolet, P., Odlyzko, A.: Singularity analysis of generating functions. SIAM J. Discrete Math. 3(2), 216–240 (1990)
25. Wilf, H.S.: The asymptotic behavior of the Stirling numbers of the first kind. J. Combin. Theory Ser. A 64(2), 344–349 (1993)
26. Gourdon, X.: Largest component in random combinatorial structures. Discrete Math. 180, 185–209 (1998)
27. Brent, R.P.: An improved Monte Carlo factorization algorithm. BIT 20(2), 176–184 (1980)
28. Knuth, D.E.: A Classical Mind: Essays in Honour of C.A.R. Hoare, pp. 247–258. Prentice Hall International (UK) Ltd., Hertfordshire (1994)

On the Sosemanuk Related Key-IV Sets

Aleksandar Kircanski and Amr M. Youssef

Concordia Institute for Information Systems Engineering
Concordia University
Montreal, Quebec, H3G 1M8, Canada
{a_kircan,youssef}@encs.concordia.ca

Abstract. SOSEMANUK is a software-based stream cipher that has passed all three stages of the ECRYPT stream cipher project and is currently a member of the eSTREAM software portfolio. In the recent works on cryptanalysis of SOSEMANUK, its relatively small inner state size of 384 bits was identified to be one of the reasons that the attacks were possible. In this paper, we show that another consequence of the small inner state size of SOSEMANUK is the existence of several classes of (K, IV), (K', IV') pairs that yield correlated keystreams. In particular, we provide a distinguisher which requires less than 2 kilobytes of data and an inner state recovery algorithm that works for two sets of key-IV pairs of expected size $\approx 2^{128}$ each. In addition, a distinguisher requiring 2^{52} keystream words is provided for another set of pairs of SOSEMANUK instances. The expected number of such key-IV pairs is 2^{192}. Although the security of SOSEMANUK is not practically threatened, the found features add to understanding of the security of the cipher and also provide the basis for an elegant attack in the fault analysis model.

1 Introduction

SOSEMANUK [5] is a fast software-oriented stream cipher that has passed all the three phases of the ECRYPT eSTREAM competition and is currently a member of the eSTREAM Profile 1 (software portfolio). It uses a 128-bit initialization vector and allows keys of either 128-bit or 256-bits, whereas the claimed security is always 128-bits. The design of SOSEMANUK is based on the SNOW 2.0 stream cipher [10] and utilizes elements of the Serpent block cipher [6]. SOSEMANUK aims to fix weaknesses of the SNOW 2.0 design and achieves better performance, notably in the cipher's initialization phase.

The preliminary analysis [5], conducted during the SOSEMANUK design process, includes the assessment of the cipher with respect to different cryptanalytic attacks such as correlation attacks, distinguishing attacks and algebraic attacks. Public analysis that followed can be divided into two threads: guess-and-determine attacks and linear attacks. In [21], Tsunoo *et al.* reported a guess-and-determine attack that requires about 2^{224} operations. The computational complexity of the latter attack was reduced to 2^{176} operations by Feng *et al.* [12]. The two recent works on linear cryptanalysis of SOSEMANUK include [16,9]. The time complexity and storage requirement of both attacks is about 2^{148}.

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 271–287, 2012.

In particular, Cho *et al.* [9] relax the amount of required keystream needed by the attack by Lee *et al.* [16] by a factor of about 2^{10} and require about $2^{135.7}$ keystream bits for the inner state recovery by linear cryptanalysis to succeed. It should be noted that the size of the inner state in SOSEMANUK was stated as a principal reason that the attacks in [21,16] were possible. In [19], an attack in the fault analysis model requiring about 6144 faults to recover the secret inner state was provided. Finally, the slide attacks on stream ciphers were introduced in [8], where Grain stream cipher was shown to be susceptible to this type of analysis.

When compared to other recent software oriented stream ciphers, such as HC-128 [23], Rabbit [7] and SNOW 2.0 [10], SOSEMANUK has the following two particularities:

(i) A comparatively small inner state of 384 bits
(ii) Delay mechanism, by which words are sent to the output every 4 steps

Property (i) stems from the SOSEMANUK design goal which aims to reduce the processor cache pressure in order to allow the execution of the cipher primarily utilizing processor cache registers, thus achieving higher speed [5]. As a consequence of having a relatively small state, a measure of defense against guess-and-determine attacks had to be introduced. Instead of releasing 32-bit words to the output as they are generated in each step, the words are accumulated and at every 4 steps, i.e., at the end of one SOSEMANUK iteration, the 4 accumulated 32-bit words are permuted, passed through a non-linear layer and finally released as a 128-bit keystream word, which resulted in increased complexity of guess-and-determine attacks. In other words, the SOSEMANUK iteration is *self-similar*, in the sense that it contains 4 identical steps.

In this paper, we examine the impact of properties (i) and (ii) on the security of SOSEMANUK. In particular, of interest are related inner states of SOSEMANUK that are preserved over time and that, as shown in this work, yield correlated keystreams. A trivial example of such a relation is full equality between the inner states. The existence of key-IV pairs that lead to related inner states follows due to property (i), more precisely, due to the ratio between the key-IV size and inner state size, given a reasonable assumption on the random behavior of 6-round cascades of the Serpent block cipher and a simple birthday paradox argument. As for particularity (ii), the self-similarity of the iteration induced by the delay mechanism gives rise to slid inner states. Again, such slid states are achievable due to the ratio between the key-IV size and the inner state size.

The results in the paper are summarized in Table 1. In particular, a distinguisher requiring 2^{51} keystream words generated by two SOSEMANUK instances with equal LFSRs is constructed. The expected size of the set of key-IV pairs leading to such related inner states is about 2^{192} pairs. Next, it is shown that there exist two distinct types of slid SOSEMANUK inner states, the difference being in the state slide distance $d \in \{1, 2\}$. Distinguishers requiring less than 2 kilobytes of data are constructed for both slid states variants. Moreover, an inner state recovery algorithm that requires a work of less than 2^{32} and $2^{47.7}$ SOSEMANUK iterations is given, for the cases $d = 1$ and $d = 2$, respectively.

Table 1. The expected number of related key-IV pairs, distinguisher and inner state recovery complexities.

Underlying inner state relation	Expected # of related key-IV pairs	Distinguisher complexity	Inner state recovery
Full equality	2^{128}	*trivial*	-
LFSR equality, (Sect. 4)	2^{192}	2^{52} words	-
Slid LFSR, Slid FSM, $d = 1$ (Sect. 5)	2^{128}	$< 2^8$ words	2^{32} time, $2^{23.8}$ space
Slid LFSR, Slid FSM, $d = 2$ (Sect. 5)	2^{128}	$< 2^8$ words	$2^{47.7}$ time, $2^{23.8}$ space
Slid LFSR	2^{192}	-	-

The two sets of key-IV pairs leading to variants of such slid inner states are expected to have about 2^{128} pairs each. Finally, the relation by which only the corresponding two LFSRs are slided is pointed out. It is presented in the last row of Table 1 and the question whether it is possible to efficiently distinguish and mount inner state recovery attacks against such instance pairs is brought up for future research.

From the perspective of fault analysis, the slide properties of Sosemanuk provided in this paper show that the security of Sosemanuk is highly sensitive to instruction skipping type of faults. Inserting such faults has been shown to be practically possible, for example by using supply voltage variation [3,4,22,20]. If the injected fault in the cryptographic device causes certain instructions to be skipped, slid Sosemanuk keystreams may be obtained and then, the practical-complexity inner state recovery algorithms provided in this paper apply.

The rest of the paper is organized as follows. In Section 2, the specification of Sosemanuk is provided. Inner state relations preserved by the iteration steps are identified and analyzed in Section 3. A distinguisher for Sosemanuk instances with identical LFSRs is given in Section 4. The inner state recovery algorithm for the case of slide Sosemanuk pairs is provided in Section 5. Finally, our conclusion is provided in Section 6.

2 The Sosemanuk Specification and Notation Conventions

The following notation will be utilized throughout the rest of the paper:

- x^i, $x^{j,\dots,i}$: i-th bit of word x and bits from i to j of word x, respectively
- $\boxplus, \times$: addition and multiplication modulo 2^{32}, respectively
- $\oplus$ bit-wise XOR
- $\lll$: left rotation defined on 32 bit values

- $w_{t,i}$: a 4-bit value defined by $w_{t,i} = z_{t,\lfloor \frac{i}{8} \rfloor}^{i \bmod 8} \; z_{t,\lfloor \frac{i}{8} \rfloor+4}^{i \bmod 8} \; z_{t,\lfloor \frac{i}{8} \rfloor+8}^{i \bmod 8} \; z_{t,\lfloor \frac{i}{8} \rfloor+12}^{i \bmod 8}$, for $t \geq 0$, $0 \leq i \leq 31$ where $z_{t,i}$ represents the i-th byte in the big-endian notation of the keystream word z_t. In other words, $w_{t,i}$ extracts the 4 bits from z_t that correspond to the application of one Serpent S-box.
- IS_t: SOSEMANUK inner state after t steps have been executed (four steps represent one full cipher iteration)

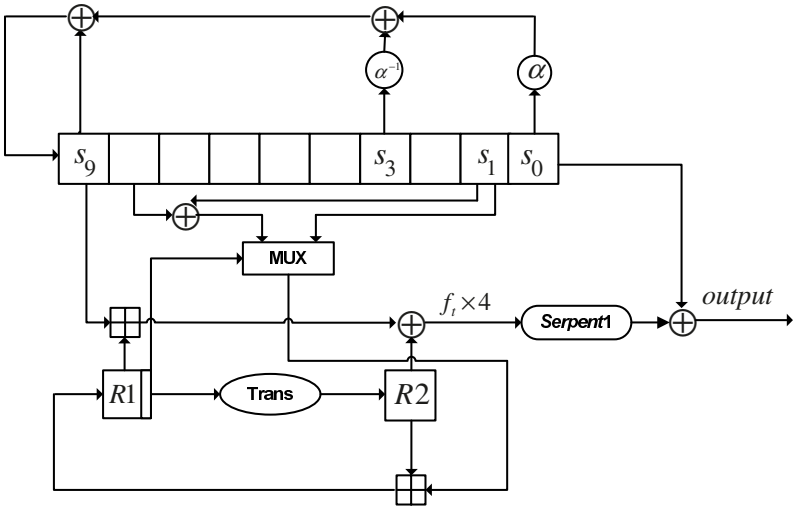

Fig. 1. Overview of the SOSEMANUK stream cipher

While the claimed security level of SOSEMANUK is 128 bits, it supports a variable key length of 128 or 256 bits, whereas the length of the initialization value can be only 128 bits. As depicted in Fig. 1, the secret inner state of SOSEMANUK consists of 12 32-bit words and utilizes three main components to generate the keystream output: a linear feedback shift register (LFSR), a finite state machine (FSM) and the function *Serpent*1 that represents the S-box layer of a Serpent round, specified using Serpent S-box S_2. The inner state at time t is denoted by $(s_t, \ldots, s_{t+9}, R1_t, R2_t)$. For notation convenience, the FSM counters in the specifications below are moved by 1 when compared to the original cipher description.

For the purpose of specifying one SOSEMANUK iteration, let α be a root of the primitive polynomial $P(X) = X^4 + \beta^{23}X^3 + \beta^{245}X^2 + \beta^{48}X + \beta^{239}$ over GF(2^8) and let β be a root of the primitive polynomial $Q(X) = X^8 + X^7 + X^5 + X^3 + 1$ over GF(2). Also, let

$$mux(c, x, y) = \begin{cases} x \text{ if } c = 0 \\ y \text{ if } c = 1 \end{cases}$$

Finally, let $Trans(x) = (M \times x) \lll 7$ and $M = 0x54655307$. A SOSEMANUK inner state update step consists of repeating the following transformations for 4 times:

$$R1_{t+1} = (R2_t \boxplus mux(lsb(R1_t), s_{t+1}, s_{t+1} \oplus s_{t+8}) \quad (1)$$

$$R2_{t+1} = Trans(R1_t) \quad (2)$$

$$s_{t+10} = s_{t+9} \oplus \alpha^{-1} s_{t+3} \oplus \alpha s_t \quad (3)$$

The multiplications by α and α^{-1} in (3) are in GF(2^{32}). Once every four steps, a 128-bit keystream word is sent to the output. In each step, between applying (2) and (3), a 32-bit f_t value is computed using

$$f_t = (s_{t+9} \boxplus R1_{t+1}) \oplus R2_{t+1} \quad (4)$$

The 128-bit keystream word is computed using

$$z_t = Serpent1(f_t f_{t+1} f_{t+2} f_{t+3}) \oplus s_t s_{t+1} s_{t+2} s_{t+3} \quad (5)$$

As for the initialization procedure of Sosemanuk, it consists of expanding the key K by the Serpent key schedule and encrypting the IV using the first 24 Serpent rounds. Denote the 128-bit i-th round output by $(Y_3^i, Y_2^i, Y_1^i, Y_0^i)$. The Sosemanuk inner state at $t = 0$ is then defined by

$$\begin{aligned}(s_6, s_7, s_8, s_9) &= (Y_3^{12}, Y_2^{12}, Y_1^{12}, Y_0^{12})\\ (s_5, R2_0, s_4, R1_0) &= (Y_3^{18}, Y_2^{18}, Y_1^{18}, Y_0^{18})\\ (s_0, s_1, s_2, s_3) &= (Y_3^{24}, Y_2^{24}, Y_1^{24}, Y_0^{24})\end{aligned}$$

where the 12-th and 18-th round outputs are taken right after the linear layer and the 24-th round output is taken after the addition with the 25-th subkey.

3 The Existence and the Size of the Related Key-IV Sets

In this section, we argue the existence of related key-IV sets and determine their respective expected sizes. First, we identify relations between the two inner states that are preserved by the Sosemanuk iteration step. A trivial example of such iteration-preserving relation is the full equality between the two inner states.

For the purpose of identifying more such relations, observe that one Sosemanuk iteration can be regarded as 4 *steps*, each consists of one FSM update followed by one LFSR update, i.e., transformations due to a single application of (1), (2) and (3). The value f_t computed in step t is preserved and at the end of the iteration, i.e., every four steps, a 128-bit keystream output value is computed using (5) and sent to the output. Note that, as for the update part, the Sosemanuk iteration is self-similar. In other words, the distance between two inner states expressed in steps is invariant with time, which makes the following definition meaningful.

Definition 1. *We say that the two inner states IS_t and IS_{t+d} (and also their corresponding cipher instances) are on distance d. Furthermore, if d is not a multiple of 4, we call such two inner states* slid *states.*

An analogous definition can be stated for distance between the two LFSRs. Now, four different iteration-preserving relations between the two inner states of SOSEMANUK can be identified:

(R1) Equality between inner states (inner states on distance $d = 0$)
(R2) Equality between LFSRs (LFSRs on distance $d = 0$)
(R3) Slid inner states (inner states on distance $1 \leq d \leq 3$)
(R4) Slid LFSRs (LFSRs on distance $1 \leq d \leq 3$)

Relations (R2) and (R4) leave the corresponding FSM content unrelated and are preserved over time since the LFSR is independent of the FSM. Relations (R1) and (R3) are trivially preserved. Note that if $d \notin \{0, 1, 2\}$, the two keystreams can be accordingly aligned, after which algorithms for (R1)-(R4) below become applicable. Since properly aligning the keystream adds more complexity to the distinguishing task, in this paper, we limit the related key-IV pair scope to only those that refer to the starting states in both instances.

To argue the existence of key-IV pairs that lead to pairs of states (R1)-(R4), let $E_K(Y^{12})$ and $E'_K(Y^{18})$ denote a composition of Serpent rounds $12, \ldots, 18$ and $18, \ldots, 24$, respectively, where E'_K also includes the addition with the 25-th round subkey. Then, it is convenient to represent the set of starting inner states with fixed value $Y^{12} = (s_6, s_7, s_8, s_9)$ by

$$\mathcal{I}(Y^{12}) = \left\{(Y^{12}, E_K(Y^{12}), E'_K(E_K(Y^{12}))) \mid K \in \{0,1\}^{128}\right\}$$

Clearly, the whole set of possible starting inner states is a union of $\mathcal{I}(Y^{12})$ sets for each $Y^{12} \in \{0,1\}^{128}$.

Consider, for example, the relation (R3) for $d = 1$. The SOSEMANUK key-IV size is 256 bits and the inner state size is 384 bits. Assume for now that E_K and E'_K behave as random functions when regarded as functions of K for fixed inputs. Such an assumption is commonly adopted in the context of time-memory tradeoff attacks [14], see also [18]. A necessary condition for the inner state IS_0 to precede state IS'_0 is that $s_{10} = s'_9$, $s_9 = s'_8$, $s_8 = s'_7$, $s_7 = s'_6$. Consider the set $\bigcup_{\alpha \in \{0,1\}^{32}} \mathcal{I}(\alpha, Y_2^{12}, Y_1^{12}, Y_0^{12})$ for a fixed Y_2^{12}, Y_1^{12}, Y_0^{12}. The set contains at most $2^{128} \times 2^{32}$ elements. After introducing a constraint that fixes s_{10} as well, the size decreases to about 2^{128} values since, due to birthday paradox, there will be only a negligible number of collisions among its elements. The only place where its potential key-IV pairs can be found is the set $\mathcal{I}(Y_2^{12}, Y_1^{12}, Y_0^{12}, s_{10})$ which contains about 2^{128} elements. The birthday paradox can be applied to the two sets using the approximation $p(m, n) \approx 1 - e^{-n^2/m}$, where m is the number of elements in the two sets and n is the number of draws from the sets [17]. By noting that $p(2^{256}, 2^{128}) \approx 0.63212$, the probability that there exists a key-IV pair that yield two slid states on $d = 1$ will be given by $1 - 0.63212^{2^{128}} \approx 1$. Analogous reasoning holds for the case when $d = 2$. Again, clearly, in case (R4), a higher existence probability is achieved both for cases $d = 1$ and $d = 2$. A simpler probabilistic argument shows that in the case of relations (R1) and (R2), related key-IV pairs almost certainly exist.

In case the Serpent 6-round cascades randomness assumption above is abandoned, the existence of SOSEMANUK related key-IV pairs is still almost certain. Assume for example that $(E_K(Y^{12}), E'_K(E_K(Y^{12})))$, regarded as a function of K, is injective in a degree significantly higher than it is the case for a randomly chosen function. In that case, the probabilistic arguments establishing relation (R1) between IS_0 and IS'_0 do not hold. However, consider the same arguments for IS_0 and IS'_n for a sufficiently large $n > 0$. Since the inner state has been randomized sufficiently by n SOSEMANUK iterations, the probabilistic arguments above can be reiterated.

The expected number of related key-IV pairs in sets specified by (R1), (R2), (R3) and (R4) is 2^{128}, 2^{192}, 2^{128} and 2^{192}, respectively. The details of the counting are given in Appendix A.

4 Sosemanuk Instances with the Identical LFSRs

In this section, we provide a distinguisher for a pair of keystreams originating from cipher instances with same LFSRs, i.e., related according to relation (R2). Since in SOSEMANUK the LFSR does not depend on the FSM registers, the equality of the LFSRs is preserved as the two instances iterate. At the first glance, it may appear that the FSM registers of the two cipher instances will become equal after about 2^{64} steps and that therefore it is trivial to distinguish the two keystreams given such long keystreams. However, as shown below, this is not the case.

Lemma 1. *The inner states of two* SOSEMANUK *instances with the same LFSRs and different FSMs will never become equal.*

Proof: It suffices to show that one SOSEMANUK step preserves the FSM difference. Whenever $R1_t \neq R1'_t$, it follows that $R2_{t+1} \neq R2'_{t+1}$, since $Trans$ is injective. If $R1_t = R1'_t$, but $R2_t \neq R2'_t$, then $R1_{t+1} \neq R1'_{t+1}$, since the mux control bits in step t are equal. □

4.1 Distinguishing a Pair of Keystreams from Random Data

Lemma 1 states that as the two instances iterate, the corresponding FSM registers will not become identical. However, as shown below, once a zero-difference is established on certain bits of the two FSMs, the equality between the bits propagates in the next steps with biased probability, yielding almost equal keystream words. The distinguisher provided below is based on this property.

The equality between the keystream words is captured by event $O_t(l)$. Namely, consider the two inner states with identical LFSRs at some time t by $(s_t, \ldots, s_{t+9}, R1_t, R2_t)$ and $(s_t, \ldots, s_{t+9}, R1'_t, R2'_t)$, where $t \equiv 0 \bmod 4$. Let the event $O_t(l)$ take place if $w_{t,0} = w'_{t,0}, \ldots, w_{t,l-1} = w'_{t,l-1}$, i.e., if the bits in the two keystream words at time t that correspond to l least significant S-boxes are equal. Below, we show that if $l > 7$, i.e., if l is larger than the rotation constant in the $Trans$ function, the probability of event $O_t(l)$ is significantly higher than the corresponding probability for randomly generated data.

In particular, the reason that the probability of $O_t(l)$ is biased when $l > 7$ can be illustrated as follows. For some fixed l and $i \in \{1, 2\}$, define

$$\Delta(Ri_t^{l-1..0}) = \begin{cases} 0 & \text{if } Ri_t^{l-1..0} = Ri_t^{'l-1..0}, \\ 1 & \text{otherwise.} \end{cases}$$

Then, if t_0 is a starting step of some SOSEMANUK iteration, the event $O_{t_0}(l)$ will take place if $(\Delta(R1_t^{l-1..0}), \Delta(R2_t^{l-1..0})) = (0, 0)$ for $t \in \{t_0, t_0 + 1, t_0 + 2, t_0 + 3\}$. Now, the mentioned bias for $l > 7$ can be observed as follows. Consider the probability $P[(0, 0) \rightarrow (0, 0)]$, i.e., the probability that the equality between the corresponding bits in the two FSMs will be preserved. Since $(\Delta(R1_t^{l-1..0}), \Delta(R2_t^{l-1..0})) = (0, 0)$, the *mux* control bits will be equal in the two instances of the cipher and therefore $R1_{t+1}^{l-1..0} = R1_{t+1}^{'l-1..0}$ will hold. On the other hand, since $Trans(x) = (M \times x \bmod 2^{32}) \lll 7$, we have that $P[R1_{t+1}^{l+7..7} = R1_{t+1}^{'l+7..7}] = 1$ and $P[R1_{t+1}^{7..0} = R1_{t+1}^{'7..0}] = 2^{-7}$. Therefore

$$P[(0, 0) \rightarrow (0, 0)] = \begin{cases} 2^{-l} & \text{if } l \leq 7 \\ 2^{-7} & \text{if } l > 7 \end{cases}$$

Thus, increasing l to greater than 7 leaves the probability above constant, which shows that equality among the least significant bits of $R1$ and $R2$ between the two instances of the cipher propagates with good probability. To have a distinguisher, l needs to be increased up to a point where the probability of $O_t(l)$ for randomly generated data is significantly smaller than for SOSEMANUK outputs. Below, the advantage of the distinguisher for $l = 14$ is shown to be sufficiently high.

Since $(\Delta(R1_t^{l-1..0}), \Delta(R2_t^{l-1..0})) = (0, 0)$ for $l = 14$ imposes two 14-bit conditions on the two inner states, the proportion of steps in which the $(0, 0)$ event takes place is 2^{-28}. Consequently, it can be shown that, in 2^{53} steps, the event $(0, 0)$ will occur more than $2^{24.9995}$ times with probability 0.98. Out of $2^{24.9995}$ $(0, 0)$ states, only $2^{22.9995}$ will represent the starting step of a SOSEMANUK iteration. Each such starting step may develop into $O_t(14)$ event with probability $P[(0, 0) \rightarrow (0, 0)]^3 = 2^{-7\times3} = 2^{-21}$. Out of $2^{22.9995}$ such steps, one will develop into $O_t(14)$ with probability $1-(1-2^{-21})^{2^{22.9995}} = 0.98$. Therefore, the probability that $O_t(14)$ will occur within 2^{53} steps, or 2^{51} SOSEMANUK iterations of the two cipher instances is higher than $0.98 \times 0.98 = 0.96$. On the other hand, the probability of that $O_t(14)$ will occur in a pair of randomly generated sequences of 2^{51} 128-bit words is $1-(1-2^{4\times14})^{2^{51}} = 0.03$, which gives rise to the following distinguisher:

- If the event $O_t(14)$ took place for some t in the two keystreams, then return SOSEMANUK. Otherwise, return RANDOM.

The overall amount of data required for the distinguisher is $2^{51} \times 2 = 2^{52}$ keystream words. The probability of false positives is 0.03 and the probability of false negatives is $1 - 0.96 = 0.04$. Tuning the number of observed keystream words as well as the value of l yields different success rates with respect to false positives, false negatives and the required data amount.

5 Sosemanuk Instances with Slid Inner States

In this section we present a practical inner state recovery algorithm, that requires a small amount of keystream generated by two instances following relation (R3) with $d = 1$ or $d = 2$. Such keystreams are easily distinguished from random keystreams given only about 32 and 64 keystream words generated by both instances for $d = 1$ and $d = 2$, respectively. The details of the distinguisher in question are provided in Appendix B.

5.1 Equations Due to Slid Sosemanuk Instances

Consider the bits that interact with one particular S-box, up to the point where they reach the output as 4 bits dispersed in the keystream word z_t. The S-boxes are applied to the input values of the form $f^i_{t+3}f^i_{t+2}f^i_{t+1}f^i_t$, for $0 \le i \le 31$ and the corresponding S-box output, after addition to the LFSR bits, represents the 4 bit output $w_{t,i} = S(f^i_{t+3}f^i_{t+2}f^i_{t+1}f^i_t) \oplus s^i_{t+3}s^i_{t+2}s^i_{t+1}s^i_t$. The particular position of the $w_{t,i}$ bits in the 128-bit keystream word is made explicit in the notation part of Section 2.

Slid state generated keystreams introduce equations that leak the inner state material. In this section, and without loss of generality, these equations are written for $t = 0$. Expand $w_{0,i}$, $w_{4,i}$, $w_{8,i}$ and $w_{12,i}$ for some $0 \le i \le 31$ using the expression $S(f^i_{t+3}f^i_{t+2}f^i_{t+1}f^i_t) \oplus s^i_{t+3}s^i_{t+2}s^i_{t+1}s^i_t = w_{t,i}$. Now, in case the 4 starting keystream words due to inner state IS_1 are also known, the system can be extended as follows

$$S(f^i_3f^i_2f^i_1f^i_0) \oplus s^i_3s^i_2s^i_1s^i_0 = w_{0,i}, \qquad S(f^i_4f^i_3f^i_2f^i_1) \oplus s^i_4s^i_3s^i_2s^i_1 = w_{1,i} \tag{6}$$

$$S(f^i_7f^i_6f^i_5f^i_4) \oplus s^i_7s^i_6s^i_5s^i_4 = w_{4,i}, \qquad S(f^i_8f^i_7f^i_6f^i_5) \oplus s^i_8s^i_7s^i_6s^i_5 = w_{5,i} \tag{7}$$

$$S(f^i_{11}f^i_{10}f^i_9f^i_8) \oplus s^i_{11}s^i_{10}s^i_9s^i_8 = w_{8,i}, \qquad S(f^i_{12}f^i_{11}f^i_{10}f^i_9) \oplus s^i_{12}s^i_{11}s^i_{10}s^i_9 = w_{9,i} \tag{8}$$

$$S(f^i_{15}f^i_{14}f^i_{13}f^i_{12}) \oplus s^i_{15}s^i_{14}s^i_{13}s^i_{12} = w_{12,i}, \quad S(f^i_{16}f^i_{15}f^i_{14}f^i_{13}) \oplus s^i_{16}s^i_{15}s^i_{14}s^i_{13} = w_{13,i} \tag{9}$$

If instead of IS_1, the keystream due to inner state IS_2 is known, the equations analogous to (6)-(9) for the case when the slide is by 2 steps are:

$$S(f^i_3f^i_2f^i_1f^i_0) \oplus s^i_3s^i_2s^i_1s^i_0 = w_{0,i}, \qquad S(f^i_5f^i_4f^i_3f^i_2) \oplus s^i_5s^i_4s^i_3s^i_2 = w_{2,i} \tag{10}$$

$$S(f^i_7f^i_6f^i_5f^i_4) \oplus s^i_7s^i_6s^i_5s^i_4 = w_{4,i}, \qquad S(f^i_9f^i_8f^i_7f^i_6) \oplus s^i_9s^i_8s^i_7s^i_6 = w_{6,i} \tag{11}$$

$$S(f^i_{11}f^i_{10}f^i_9f^i_8) \oplus s^i_{11}s^i_{10}s^i_9s^i_8 = w_{8,i}, \qquad S(f^i_{13}f^i_{12}f^i_{11}f^i_{10}) \oplus s^i_{13}s^i_{12}s^i_{11}s^i_{10} = w_{10,i} \tag{12}$$

$$S(f^i_{15}f^i_{14}f^i_{13}f^i_{12}) \oplus s^i_{15}s^i_{14}s^i_{13}s^i_{12} = w_{12,i}, \quad S(f^i_{17}f^i_{16}f^i_{15}f^i_{14}) \oplus s^i_{17}s^i_{16}s^i_{15}s^i_{14} = w_{14,i} \tag{13}$$

The problem is how to recover the inner state given the right-hand values of (6)-(9) or (10)-(13), for each bit position $i = 0, \ldots, 31$.

5.2 Recovering the Inner State

The inner state is recovered by first recovering the LFSR and then the FSM. To recover the LFSR, first, equations (6)-(9) or (10)-(13) are solved, yielding a

constraint on the extended LFSR (i.e., $s_0, \ldots, s_{16}$.) Note that the equations have been deliberately chosen to cover variables of the extended LFSR, rather than the LFSR. That way, the dependence between the LFSR registers expressed by (3) can be used to further restrict the LFSR space. Namely, having a restriction against $s_0, \ldots, s_{16}$ in case of system (6)-(9), instead of against only $s_0, \ldots, s_9$, allows pruning the wrong $s_0, \ldots, s_9$ candidates by using (3) for $t = 0, \ldots, 6$.

To find the expected number of solutions to the system (6)-(9), the exact distribution of the number of solutions was calculated. The problem of calculating the distribution was divided into two parts. The distribution was found first for (6)-(7) and then for (8)-(9), where care has been taken about bits s_8 and f_8 which participate in both subsystems. Once the joint distribution that represents the distribution of the number of solutions of the full system (6)-(9) was calculated, the expected number of solutions for the system was found to be $2^{3.357}$. The same strategy has been applied for the case $d = 2$, and the expected number of solutions to (10)-(13) is found to be equal to $2^{4.5278}$. Since the 32 systems corresponding to different $0 \leq i \leq 31$ are independent, the overall number of candidates for $(s_0, \ldots, s_{16})$ is expected to be $2^{3.357 \times 32} \approx 2^{107.42}$ and $2^{32 \times 4.528} \approx 2^{144.896}$ for $d = 1$ and $d = 2$, respectively. As for mutual dependencies between $(s_0, \ldots, s_{16})$, relation (3) for $t = 0, \ldots, 6$ imposes 7 32-bit constraints on the extended LFSR. Therefore, the two restrictions ensure that the final number of candidates for the extended LFSR is restricted only to 1 (correct) value with high probability both in cases $d = 1$ and $d = 2$, since $7 \times 32 = 224 >> 107.42$ and $7 \times 32 = 224 >> 144.896$, respectively.

Recovering the LFSR To recover the LFSR in case $d = 1$, the attacker obtains the first 4 keystream words of both instances and solves 32 systems of the form (6)-(9), by simply trying all the possibilities for the $f_0^i, \ldots, f_{16}^i$ and whenever possible, deducing the corresponding $s_0^i, \ldots, s_{16}^i$. The resulting extended LFSR restriction is specified "i-th-bit-wise", that is to say, the solution of the 32 systems can be regarded as sets $S_0, \ldots, S_{31}$ where each set S_i contains about $2^{3.357}$ 17-bit values as candidates for value $s_0^i \ldots, s_{16}^i$. To prune the false candidates, a naive approach of going through all candidates of the 32 sets and discarding in case the α relations are not satisfied would require an impractical complexity of $2^{108.16}$ operations. A more efficient approach is to utilize the properties of the α and α^{-1} multiplication in $GF(2^{32})$ in order to derive an early contradiction after choosing candidates from $S_{l_1}, \ldots, S_{l_m}$ for m significantly smaller than 32.

Multiplication by α and α^{-1} in GF(2^{32}) can be represented as [5]:

$$\alpha(x) = (x \gg 8) \oplus T_\alpha(x\&255), \quad \alpha^{-1}(x) = (x \ll 8) \oplus T_\beta(x \gg 24) \tag{14}$$

where T_α and T_β represent 8×32 S-boxes. Let $s_{t,b}$, $0 \leq b \leq 3$ denote the b-th least significant byte of the LFSR word s_t. Isolating the i-th bit of each value participating in relation (3) and using (14), the following bit relation is obtained

$$s_{t+10}^i = s_{t+9}^i \oplus s_{t+3}^{i-8} \oplus T_\beta^i(s_{t+3,3}) \oplus s_t^{i+8} \oplus T_\alpha^i(s_{t,0}) \tag{15}$$

with the convention that bits s_{t+3}^{i-8} for $0 \leq i \leq 7$ and s_t^{i+8} for $24 \leq i \leq 31$ are equal to 0. Choosing a candidate from the set S_i for some $0 \leq i \leq 31$ produces one such equation for each $0 \leq t \leq 6$, i.e., 7 equations of form (15). In each equation, due to the candidate choice, s_{t+10}^i and s_{t+9}^i are known bits. No two equations among the 7 equations have common variables. More generally, choosing a candidate from each of the m different S_i sets for some $0 \leq m \leq 31$, generates 7 independent systems, each having m equations of the form (15). Since increasing the number of sets S_i from which the candidates are chosen enlarges the search space, the strategy is to select a minimal family of S_i sets so that each of the systems is solvable in $s_{t+3,3}$ and $s_{t,0}$, the two input bytes for T_α and T_β S-boxes and moreover, so that the system becomes overdefined. Then, in case the solution to any of the systems does not exist, the candidates choice from the S_i sets can be discarded.

The strategy above is illustrated as follows. Let $i = 0$ and let a candidate be chosen from S_0. Note that (15) for $i = 0$ contains the unknown variable s_t^8. Since the goal is to make the systems solvable for T_α and T_β S-box inputs, in order to fix s_8, a candidate from S_8 is chosen. Similarly, other enforced choices are from sets S_{16} and S_{24}. For a fixed $0 \leq t \leq 6$, the system is of the form

$$\begin{aligned} s_{t+10}^0 &= s_{t+9}^0 \oplus s_t^8 \oplus T^0(s_{t+3,3}, s_{t,0}) \\ s_{t+10}^8 &= s_{t+9}^8 \oplus s_{t+3}^0 \oplus s_t^{16} \oplus T^8(s_{t+3,3}, s_{t,0}) \\ s_{t+10}^{16} &= s_{t+9}^{16} \oplus s_{t+3}^8 \oplus s_t^{24} \oplus T^{16}(s_{t+3,3}, s_{t,0}) \\ s_{t+10}^{24} &= s_{t+9}^{24} \oplus s_{t+3}^{16} \oplus T^{24}(s_{t+3,3}, s_{t,0}) \end{aligned} \tag{16}$$

where $T(s_{t+3,3}, s_{t,0}) = T_\beta(s_{t+3,3}) \oplus T_\alpha(s_{t,0})$ is a 16×32 one-to-one S-box (as we verified) convenient to define for the consideration below. Given the system (16), the bits $T^0(s_{t+3,3}, s_{t,0})$, $T^8(s_{t+3,3}, s_{t,0})$, $T^{16}(s_{t+3,3}, s_{t,0})$ and $T^{24}(s_{t+3,3}, s_{t,0})$ can be determined. Moreover, the choice of candidates from S_0 and S_{24} imposes a constraint on the T input by fixing the least significant bit of $s_{t,0}$ and $s_{t+3,0}$, respectively. The same holds for any i, $0 \leq i \leq 7$. Thus, the system constraints the T input-output value by fixing its 4 output and 2 input bits.

Similarly, for $0 \leq i < 8$, choosing elements from S_{i+8k}, $k = 0, 1, 2, 3$, yields a system analogous to (16), for each $0 \leq t \leq 6$. Thus, the T input-output is constrained by fixing 2 input and 4 output bits, for each $0 \leq t \leq 6$. Now, extending the guess to sets $S_{i_0+8k}, \ldots, S_{i_n+8k}$, for $k = 0, 1, 2, 3$ and some $0 \leq i_0, \ldots, i_n \leq 7$ will overdefine the T input with good probability if $n \geq 3$. In particular, for $n = 3$, each system constraints the T input value by fixing 6 input bits and 12 output bits. Since there exists only 2^{16} possible input-output pairs for T, at most 2^{16} out of 2^{18} such constraints will be satisfiable. As it is reasonable to assume that the constraint values are distributed uniformly, the candidate choice will pass with probability of at most 2^{-2} for one fixed $0 \leq t \leq 6$. Then, the pruning is done by verifying whether for the given choice from S_{i_0+8k}, S_{i_1+8k} and S_{i_2+8k}, for $k = 0, 1, 2, 3$ all of the 7 systems are satisfiable, which happens with probability less than $2^{-2\times 7} = 2^{-14}$. Since the initial number of candidates is expected to be $2^{12\times 3.357} = 2^{40.284}$, less than $2^{40.56-14} = 2^{26.56}$ candidates

are expected to pass the test. In case $n = 4$, using analogous arguments, each of the 7 16-equation systems is satisfiable with probability of at most 2^{-8} and thus applying this criterion provides reduction of at least $2^{-8\times 7} = 2^{-56}$. The initial expected number of candidates in this case is $2^{3.357\times 16} = 2^{53.712}$ and thus no false candidates are expected to pass the test. It can thus be observed that for $n = 3$, the criterion is weaker but there is a smaller number of candidates, whereas for $n = 4$, the criterion is stronger, but the initial number of candidates is relatively high. Note that above values $i_0, \ldots, i_3$ can be chosen arbitrarily. However, to decrease the complexity of the search, it is useful to start from i_j values such that $|S_{i_j}|$ are small.

The discussion above indicates a breadth-first style search procedure to efficiently prune incorrect candidates. Namely, given the solution to (6)-(9), in the form of sets $S_0, \ldots, S_{31}$, the LFSR recovery proceeds as follows. Let $h(i) = \prod_{0\le k\le 3} |S_{i+8k}|$. Determine $i_0, \ldots, i_7$ so that $h(i_0), h(i_1), \ldots, h(i_7)$ represents a sequence sorted in increasing order. Choose a candidate from each of the S_{i_0+8k}, S_{i_1+8k} and S_{i_2+8k}, $0 \le k \le 3$. Generate 7 systems, 12 equations of form (15) each, using the guessed candidates. If for any of the 7 systems, the constrained T input-output value does not exist, discard the candidates choice. Otherwise, for the candidates that passed the criterion, continue the search by guessing the next four S_{i_j+8k} sets simultaneously and looking for contradiction in a breadth-first manner, i.e., not by recursing into the search tree, but rather looking for contradiction after every fourplet guess. Note that solving the systems appearing during the attack can be done efficiently by utilizing precomputed tables. For some $0 \le i_0, i_1, i_2 \le 7$, consider verifying whether a guess from sets S_{i_0+8k}, S_{i_1+8k} and S_{i_2+8k}, $0 \le k \le 3$ yields a contradiction or not. Each such guess yields an 18-bit constraint on the input-output value of T, where the set of fixed bit positions uniquely depends on the set $\{i_0, i_1, i_2\}$. A yes/no information on the existence of such T input-output value can be preserved in precomputed tables $T_{\{i_0,i_1,i_2\}}$, for each set $\{i_0, i_1, i_2\} \in \{0, 1, \ldots, 7\}$. The storage demand amounts to $\binom{8}{3} \times 2^{18} \approx 2^{23.8}$ bits and the same number of T lookup operations is required to fill in the tables.

Estimated Complexities and Experimental Results. In the previous section, a procedure for recovering the LFSR was provided. After the LFSR is recovered, the FSM can be recovered by guessing one FSM register, then recovering the other one and verifying the guess against the keystream. The computational effort of such a procedure is less than 2^{32} SOSEMANUK iterations. According to the previous subsection, the computational effort of the LFSR recovery amounts to the search over candidates from sets S_{i_0+8k}, S_{i_1+8k} and S_{i_2+8k}, for $0 \le k \le 3$, where $0 \le i_0, i_1, i_2 \le 7$ are the three points of the function $h(i) = \prod_{0\le k\le 3} |S_{i+8k}|$ with the smallest values. To estimate the expected value of such three smallest values, for both $d = 1$ and $d = 2$, the exact distribution of $h(i)$ was calculated exhaustively, using the exact distribution of $|S_{i+8k}|$, found at the beginning of this section. The attacker has a chance to choose the three smallest points of h, out of 8 independent $h(i)$ for $0 \le i < 8$. The 1st, 2nd and 3rd order statistics of

an 8 element sample has been found to be $2^{9.517}$, $2^{10.602}$ and $2^{11.372}$, respectively. Therefore, the overall attack requirements for $d = 1$ are

- Data of 4 words, produced by 2 slid SOSEMANUK instances
- The computation of $2^{9.517+10.602+11.372} = 2^{31.491}$ table lookups for the LFSR recovery and the work equivalent to 2^{32} iterations for the FSM recovery
- Storage requirement of $2^{23.8}$ bits

For $d = 2$, the data and storage complexity remain the same. Since the expected 1^{st}, 2^{nd} and 3^{rd} order statistics of $h(i)$, $0 \leq i \leq 7$, are $2^{15.081}$, $2^{15.996}$ and $2^{16.627}$, respectively, the overall expected number of candidates to go through is $2^{15.081+15.996+16.627} = 2^{47.704}$. Note that the computational cost can be lowered at the cost of increasing the available data. Namely, if more than 4 keystream words are available, the attack can be performed using the 4 consecutive keystream words of both instances for time t where $h(i_0)$, $h(i_1)$, $h(i_2)$ are minimal.

The attack was implemented for $d = 1$ on a 2.4 GHz AMD processor, using the Java SOSEMANUK implementation provided with the cipher specification. The search procedure behaved as predicted above and the two slid SOSEMANUK inner states have been uniquely recovered in less than a day.

Slide Sosemanuk instances in the context of fault attacks The properties exposed in this section yield a fault attack, in which the attacker introduces instruction skipping faults [15], e.g., by varying the supply voltage of the device that performs encryption at carefully chosen times. In such a setting, transient faults start occurring in a relatively large number [4] and the attacker waits until the slid keystreams are detected, using the distinguisher from Section 5. At that point, the inner state recovery procedure provided above applies and the secret inner state of the cipher can be efficiently recovered.

6 Conclusion and Future Work

This work provides the first steps in the analysis of the related keys of SOSEMANUK. We showed that SOSEMANUK's related key pairs exist and that observing pairs of keystreams generated under such related key-IV pairs can lead to the recovery of the inner state. In particular, an efficient inner state recovery algorithm and also a distinguisher have been provided for two particular classes of expected size of 2^{128} key-IV pairs. A distinguisher requiring 2^{52} keystream words was presented for a larger class of related key-IV pairs, of size 2^{192}. While the results from this paper do not directly threaten the security of SOSEMANUK, the non-random properties of SOSEMANUK should be brought to light and the users of the cipher should be aware of it. Finally, this work also shows that SOSEMANUK is highly sensitive to a specific kind of a fault analysis attack in which the execution flow is disturbed.

As for future work, the natural next step is to find an efficient procedure for constructing particular $((K, IV), (K', IV'))$ pairs that yield the presented related SOSEMANUK instances.

References

1. Ahamadi, H., Eghidos, T., Khazaei, S.: Improved Guess and Determine Attack on Sosemanuk, Tehran (2006), http://www.ecrypt.eu.org/stream/sosemanukp3.html
2. 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)
3. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. Proceedings of the IEEE 94(2), 370–382 (2006)
4. Barenghi, A., Bertoni, G., Breveglieri, L., Pellicioli, M., Pelosi, G.: Low Voltage Fault Attacks to AES and RSA on General Purpose Processors, ePrint IACR Report, 130/2010
5. Berbain, C., Billet, O., Canteaut, A., Courtois, N., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Minier, M., Pornin, T., Sibert, H.: Sosemanuk, a Fast Software-Oriented Stream Cipher. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 98–118. Springer, Heidelberg (2008)
6. Biham, E., Anderson, R., Knudsen, L.R.: Serpent: A New Block Cipher Proposal. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 222–238. Springer, Heidelberg (1998)
7. Boesgaard, M., Vesterager, M., Pedersen, T., Christiansen, J., Scavenius, O.: Rabbit: A New High-Performance Stream Cipher. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 307–329. Springer, Heidelberg (2003)
8. De Cannière, C., Küçük, Ö., Preneel, B.: Analysis of Grain's Initialization Algorithm. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 276–289. Springer, Heidelberg (2008)
9. Cho, J.Y., Hermelin, M.: Improved Linear Cryptanalysis of SOSEMANUK. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 101–117. Springer, Heidelberg (2010)
10. Ekdahl, P., Johansson, T.: A New Version of the Stream Cipher SNOW. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 47–61. Springer, Heidelberg (2003)
11. eSTREAM, the ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream/
12. Feng, X., Liu, J., Zhou, Z., Wu, C., Feng, D.: A Byte-Based Guess and Determine Attack on SOSEMANUK. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 146–157. Springer, Heidelberg (2010)
13. Grinstead, C.M., Snell, L.J.: Introduction to Probability. American Mathematical Society, 2nd edn. (1998)
14. Hellman, M.: A Cryptanalytic Time-Memory Trade-Off. IEEE Transactions on Information Theory IT-26, 401–406 (1980)
15. Kim, C.H., Quisquater, J.-J.: Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) WISTP 2007. LNCS, vol. 4462, pp. 215–228. Springer, Heidelberg (2007)
16. Lee, J.-K., Lee, D.-H., Park, S.: Cryptanalysis of Sosemanuk and SNOW 2.0 Using Linear Masks. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 524–538. Springer, Heidelberg (2008)
17. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)

18. Quisquater, J.-J., Delescaille, J.-P.: How Easy Is Collision Search? Application to DES. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 429–434. Springer, Heidelberg (1990)
19. Esmaeili Salehani, Y., Kircanski, A., Youssef, A.: Differential Fault Analysis of SOSEMANUK. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 316–331. Springer, Heidelberg (2011)
20. Schmidt, J.-M., Herbst, C.: A Practical Fault Attack on Square and Multiply. In: Fault Diagnosis and Tolerance in Cryptography, 3rd International Workshop, FDTC 2008. IEEE-CS Press (2008)
21. Tsunoo, Y., Saito, T., Shigeri, M., Suzaki, T., Ahmadi, H., Eghlidos, T., Khazaei, S.: Evaluation of SOSEMANUK With Regard to Guess-and-Determine attacks (2006), http://www.ecrypt.eu.org/stream/soemanukp3.html
22. Vertanen, O.: Java Type Confusion and Fault Attacks. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 237–251. Springer, Heidelberg (2006)
23. Wu, H.: The Stream Cipher HC-128. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 39–47. Springer, Heidelberg (2008)

A Estimating the Expected Number of Related Key-IV Pairs

Having shown the almost certain existence of related key-IV pairs in Section 3, in this appendix, we estimate their corresponding number. This is done for each relation (R1)-(R4), under the assumption that $(E_K(Y^{12}), E'_K(E_K(Y^{12})))$ behaves randomly when regarded as functions of the key. Note that if m elements are drawn with replacement from a set of n elements, the expected number of drawn elements that will repeat is $E_1(m,n) = \binom{m}{2}\frac{1}{n}$, since the probability that two elements will collide is $\frac{1}{n}$ and there exist $\binom{m}{2}$ elements pairs. Similarly, if two sets of m elements are drawn from an n-element set, the expected number of collisions is $E_2(m,n) = \frac{m^2}{n}$.

As for case (R1), for some $Y^{12} \in \{0,1\}^{128}$, the expected number of collisions in $\mathcal{I}(Y^{12})$ is $E_1(2^{128}, 2^{256}) \approx 1$. Since there is 2^{128} possibilities for Y^{12}, we have that the final expected number of key-IV pairs that yield identical inner states is $2^{128} \times E_1(2^{128}, 2^{256}) \approx 2^{128}$. In case (R2), since only the LFSRs are required to be identical, we have that the expected number of key-IV pairs is $2^{128} \times E_1(2^{128}, 2^{192}) \approx 2^{192}$.

As for slid pairs, described by case (R3), let $Y_2^{12}, Y_1^{12}, Y_0^{12}$ be fixed. Consider the subset of the set $\bigcup_{\alpha \in \{0,1\}^{32}} \mathcal{I}(\alpha, Y_2^{12}, Y_1^{12}, Y_0^{12})$ that fixes the corresponding s^{10} value. As described in the previous subsection, such a set is expected to have 2^{128} elements. The corresponding slid pairs can only be in $\mathcal{I}(Y_2^{12}, Y_1^{12}, Y_0^{12}, s_{10})$. Since the two sets have 2^{128} elements, and there is 2^{128} such set pairs, the expected number of states that have a corresponding slid state is $2^{128} \times E_2(2^{128}, 2^{256}) \approx 2^{128}$. Even though when $\alpha = Y_2^{12} = Y_1^{12} = Y_0^{12}$ there may be an overlap between the corresponding sets, it is easy to show that the final expected number of slid pairs does not change significantly in this case. Applying analogous reasoning, it follows that the number of key-IV pairs corresponding to case (R4) is $2^{128} \times E_2(2^{128}, 2^{192}) \approx 2^{192}$.

B Distinguishing Slid Sosemanuk Instances

In this appendix, we show that the output of two SOSEMANUK instances on distance $d = 1$ or $d = 2$ can easily be distinguished from random stream. Let $z_0, z_4, \ldots, z_{4n}$ and $z_1, z_5, \ldots, z_{4n+1}$, be two SOSEMANUK keystream outputs generated by slid inner state on distance $d = 1$. First, we show that it is possible to efficiently compute the probability that the two slid SOSEMANUK instances generate a given sequence $z = (z_0, z_1, z_4, z_5, \ldots, z_{4n}, z_{4n+1})$.

Assume that sequence z is due to two slid SOSEMANUK instances and extract the S-box outputs corresponding to different bit positions $0 \leq i \leq 31$. Write down each of the 32 subsequences in the form

$$w_{0,i}, w_{1,i}, w_{4,i}, w_{5,i}, w_{8,i}, w_{9,i}, \ldots, w_{4n,i}, w_{4n+1,i} \tag{17}$$

The first 8 elements in the sequence (17) are generated according to (6)-(9) and the rest are generated by analogous equations. Consider the dependence between the elements of the sequence (17) by focusing for example on the value $w_{4,i}$ defined by the left-hand equation of (7). Observe that values participating in its generation $f_4, \ldots, f_7$ and $s_4, \ldots, s_7$ participate only in the generation of $w_{1,i}$ and $w_{5,i}$. The same observation generalizes to any sequence element in (17). We adopt the assumption that f_t and s_t, $t \geq 0$, are uniformly distributed. Although the assumption does not fully ressemble the real case, it serves to simplify the model and moreover, does not create any relevant deviation from the real case, as verified by the experimental results below. Now each value in (17) depends only on its predecessor. Thus, the behavior of the subsequence is completely determined by transition probabilities, which are different for even and odd steps of the sequence (17). As for the transitions of the form $w_{4t,i}$ to $w_{4t+1,i}$, the transition matrix $M_{1,1}$ is provided. The elements of $M_{1,1}$ that represent transition probabilities have been calculated by going through all of the left-hand values of a pair of equations (6) and recording the number of transitions for each $(w_{0,i}, w_{1,i})$:

$$M_{1,1} = \frac{1}{64} \begin{pmatrix}
4 & 4 & 3 & 5 & 3 & 5 & 6 & 2 & 4 & 4 & 3 & 5 & 3 & 5 & 6 & 2 \\
4 & 4 & 3 & 5 & 3 & 5 & 6 & 2 & 4 & 4 & 3 & 5 & 3 & 5 & 6 & 2 \\
4 & 4 & 5 & 3 & 5 & 3 & 2 & 6 & 4 & 4 & 5 & 3 & 5 & 3 & 2 & 6 \\
4 & 4 & 5 & 3 & 5 & 3 & 2 & 6 & 4 & 4 & 5 & 3 & 5 & 3 & 2 & 6 \\
3 & 5 & 4 & 4 & 6 & 2 & 3 & 5 & 3 & 5 & 4 & 4 & 6 & 2 & 3 & 5 \\
3 & 5 & 4 & 4 & 6 & 2 & 3 & 5 & 3 & 5 & 4 & 4 & 6 & 2 & 3 & 5 \\
5 & 3 & 4 & 4 & 2 & 6 & 5 & 3 & 5 & 3 & 4 & 4 & 2 & 6 & 5 & 3 \\
5 & 3 & 4 & 4 & 2 & 6 & 5 & 3 & 5 & 3 & 4 & 4 & 2 & 6 & 5 & 3 \\
3 & 5 & 6 & 2 & 4 & 4 & 3 & 5 & 3 & 5 & 6 & 2 & 4 & 4 & 3 & 5 \\
3 & 5 & 6 & 2 & 4 & 4 & 3 & 5 & 3 & 5 & 6 & 2 & 4 & 4 & 3 & 5 \\
5 & 3 & 2 & 6 & 4 & 4 & 5 & 3 & 5 & 3 & 2 & 6 & 4 & 4 & 5 & 3 \\
5 & 3 & 2 & 6 & 4 & 4 & 5 & 3 & 5 & 3 & 2 & 6 & 4 & 4 & 5 & 3 \\
6 & 2 & 3 & 5 & 3 & 5 & 4 & 4 & 6 & 2 & 3 & 5 & 3 & 5 & 4 & 4 \\
6 & 2 & 3 & 5 & 3 & 5 & 4 & 4 & 6 & 2 & 3 & 5 & 3 & 5 & 4 & 4 \\
2 & 6 & 5 & 3 & 5 & 3 & 4 & 4 & 2 & 6 & 5 & 3 & 5 & 3 & 4 & 4 \\
2 & 6 & 5 & 3 & 5 & 3 & 4 & 4 & 2 & 6 & 5 & 3 & 5 & 3 & 4 & 4
\end{pmatrix}$$

The factor $\frac{1}{64}$ comes from the fact that fixing $w_{0,i}$ in (6) leaves 2^4 possibilities for f_0, f_1, f_2, f_3, s_0, s_1, s_2 and s_3 and thus $2^{4+2} = 64$ possibilities for the left-hand side of the equation for $w_{1,i}$. From matrix $M_{1,1}$ it is clear that the transition probabilities are heavily biased.

In the case of transitions from $w_{4t+1,i}$ to $w_{4(t+1),i}$, the bias does not exist and all the values in the 16×16 matrix $M_{1,2}$ are equal to $\frac{1}{16}$. Now the probability $\Pr_{D_0}[z]$, where D_0 is the distribution that describes the output of two slid SOSEMANUK instances, can be calculated efficiently. Namely, given a reasonable independence assumption, this probability is equal to a product of probabilities of 32 subsequences of form (17) extracted from z. Each of the subsequence probabilities is calculated by using the Markovian assumption and the transition matrices $M_{1,1}$, $M_{1,2}$. On the other hand, the probability $\Pr_{D_1}[z]$ of z occurring in a random stream is equal to $1/2^{(2n+2)\times 128}$.

Finally, as for the matrix $M_{2,1}$ that corresponds to the case $d = 2$, this 16×16 matrix has been found to be populated only by $\frac{13}{256}$ and $\frac{19}{256}$ values. Denote the elements of the matrix in question by $M_{2,1} = [a_{i,j}]$. Then, for $i \in \{1, 2, 3, 4, 13, 14, 15, 16\}$, $a_{i,j} = \frac{13}{256}$ if $j \in \{1, 4, 5, 8, 9, 12, 13, 16\}$. For $i \in \{5, 6, 7, 8, 9, 10, 11, 12\}$, $a_{i,j} = \frac{13}{256}$ if $j \in \{2, 3, 6, 7, 10, 11, 14, 15\}$. Other $a_{i,j}$ elements are equal to $\frac{19}{256}$.

To optimally decide whether the sample z comes from distribution D_0 or D_1 [2], the acceptance region is defined by

$$\mathcal{A} = \{z : \mathrm{LR}(z) \geq 1\}, \text{ where } \mathrm{LR}(z) = \frac{\Pr_{D_0}[z]}{\Pr_{D_1}[z]}$$

If $z \in \mathcal{A}$, the distinguisher returns D_0 and otherwise D_1. Thus, to distinguish outputs of two SOSEMANUK slid instances from random data, it suffices to compute the probability of the sample under distributions D_0 and D_1 and compare the two probabilities.

The distinguisher above has been implemented and the following experiments have been performed to estimate the number of keystream words necessary for the overall probability of error P_e to be sufficiently low. Each experiment consisted of generating 32 keystream words by each of the two random instances of SOSEMANUK which are on distance 1 and then applying the distinguisher, whereas the probability multiplicative structure has been replaced by the usual logarithmic additive representation. The experiment was repeated for 1024 times and no false negatives or false positives have been reported. It can be concluded that the input size of 32 keystream words is sufficient to distinguish SOSEMANUK with high probability. The analogous experiment was executed for the case $d = 2$ using 64 keystream words from both instances. Neither false positives nor false negatives occurred in 1024 experiments repetitions, which shows that 64 keystream words are sufficient to make P_e sufficiently small when $d = 2$.

High Speed Implementation of Authenticated Encryption for the MSP430X Microcontroller

Conrado P.L. Gouvêa* and Julio López

University of Campinas (Unicamp)
{conradoplg,jlopez}@ic.unicamp.br

Abstract. Authenticated encryption is a symmetric cryptography scheme that provides both confidentiality and authentication. In this work we describe an optimized implementation of authenticated encryption for the MSP430X family of microcontrollers. The CCM, GCM, SGCM, OCB3, Hummingbird-2 and MASHA authenticated encryption schemes were implemented at the 128-bit level of security and their performance was compared. The AES accelerator included in some models of the MSP430X family is also studied and we explore its characteristics to improve the performance of the implemented modes, achieving up to 10 times of speedup. The CCM and OCB3 schemes were the fastest when using the AES accelerator while MASHA and Hummingbird-2 were the fastest when using only software.

Keywords: authenticated encryption, MSP430, AES, software implementation.

1 Introduction

Constrained platforms such as sensor nodes, smart cards and radio-frequency identification (RFID) devices have a great number of applications, many of which with security requirements that require cryptographic schemes. The implementation of such schemes in these devices is very challenging since it must provide high speed while consuming a small amount of resources (energy, code size and RAM). In this scenario, symmetric cryptography becomes an essential tool in the development of security solutions, since it can provide both confidentiality and authenticity after being bootstrapped by some protocol for key agreement or distribution. Encryption and authentication can be done through generic composition of separate methods; however, the study of an alternative approach named authenticated encryption (AE) has gained popularity.

Authenticated encryption provides both confidentiality and authenticity within a single scheme. It is often more efficient than using separate methods and usually consumes a smaller amount of resources. It also prevents common critical mistakes when combining encryption and authentication such as not using separate keys for each task. There are many AE schemes; see e.g. [10] for a non-exhaustive list. Some AE schemes are built using a block cipher, in this case, they

* Supported by FAPESP, grant 2010/15340-3.

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 288–304, 2012.

are also called AE modes. In this work, we follow the approach from [10] and compare the Counter with CBC-MAC (CCM) mode [18], the Galois/Counter Mode (GCM) [13] and the Offset Codebook (OCB3) mode [10]. We have also implemented the Sophie Germain Counter Mode [14], the Hummingbird-2 cipher [5] and the MASHA cipher [9]. The CCM mode and GCM have been standardized by the National Institute of Standards and Technology (NIST); CCM is used for Wi-Fi WPA2 security (IEEE 802.11i) while GCM is used in TLS, IPSec and NSA Suite B, for example. The recently proposed OCB3 mode is the third iteration of the OCB mode and appears to be very efficient in multiple platforms. The SGCM is a variant of GCM and was proposed to be resistant against some existing attacks against GCM while being equally or more efficient; we have implemented it in order to check this claim and compare it to GCM. The Hummingbird-2 cipher (which may be referred to as HB2 in this work) is specially suited for 16-bit platforms and was implemented in order to compare it to the other non-specially suited modes. The MASHA cipher is based on a stream cipher and claims to fill the gap for authenticated encryption algorithms based on stream ciphers which achieve a good balance between security and performance.

The goal of this work is to provide an efficient implementation and comparison of the aforementioned AE schemes (CCM, GCM, SGCM, OCB3, Hummingbird-2, and MASHA) for the MSP430X microcontroller family from Texas Instruments. This family is an extension of the MSP430 which have been used in multiple scenarios such as wireless sensor networks; furthermore, some microcontrollers of this family feature an AES accelerator module which can encrypt and decrypt using 128-bit keys. Our main contributions are: (i) to study (for the first time, to the best of our knowledge) the efficient usage and impact of this AES accelerator module in the implemented AE schemes; (ii) to describe a high speed implementation of those AE schemes for the MSP430X, achieving performance 10 times faster for CCM using the AES accelerator instead of AES in software; (iii) to describe an efficient implementation of AES for 16-bit platforms; (iv) to show that CCM is the fastest of those schemes whenever a non-parallel AES accelerator is available; and (v) and to provide a comparison of the six AE schemes, with and without the AES accelerator. We remark that the results regarding the efficient usage of the AES accelerator can be applied to other devices featuring analogue accelerators, such as the AVR XMEGA.

This paper is organized as follows. In Section 2, the MSP430X microcontroller family is described. Section 3 offers an introduction to AE. Our implementation is described in Section 4, and the obtained results are detailed in Section 5. Section 6 provides concluding remarks.

2 The MSP430X Family

The MSP430X family is composed by many microcontrollers which share the same instruction set and 12 general purpose registers. Although it is essentially a 16-bit architecture, its registers have 20 bits, supporting up to 1 MB of addressing space. Each microcontroller has distinct clock frequency, RAM and flash sizes.

Some MSP430X microcontrollers (namely the CC430 series) have an integrated radio frequency transceiver, making them very suitable for wireless sensors. These models also feature an AES accelerator module that supports encryption and decryption with 128-bit keys only. The study of this accelerator is one key aspect of this study and for this reason we describe its basic usage as follows. In order to encrypt a block of 16 bytes, a flag must be set in a control register to specify encryption and the key must be written sequentially (in bytes or words) in a specific memory address. The input block must then be written, also sequentially, in another memory address. After 167 clock cycles, the result is ready and must be read sequentially from a third address. It is possible to poll a control register to check if the result is ready. Further blocks can be encrypted with the same key without writing the key again. The decryption follows the same procedure, but it requires 214 clock cycles of processing. It is worth noting that these memory read and writes are just like regular reads and writes to the RAM, and therefore the cost of communicating with the accelerator is included in our timings.

3 Authenticated Encryption

An authenticated encryption scheme is composed of two algorithms: authenticated encryption and decryption-verification (of integrity). The authenticated encryption algorithm is denoted by the function $\mathcal{E}_K(N, M, A)$ that returns (C, T), where $K \in \{0,1\}^k$ is the k-bit key, $N \in \{0,1\}^n$ is the n-bit nonce, $M \in \{0,1\}^*$ is the message, $A \in \{0,1\}^*$ is the associated data, $C \in \{0,1\}^*$ is the ciphertext and $T \in \{0,1\}^t$ is the authentication tag. The decryption-verification algorithm is denoted by the function $\mathcal{D}_K(N, C, A, T)$ that returns (M, V) where K, N, C, A, T, M are as above and V is a boolean value indicating if the given tag is valid (i.e. if the decrypted message and associated data are authentic).

Many AE schemes are built using a block cipher such as AES. Let $E_K(B)$ denote the block cipher, where the key K is usually the same used in the AE mode and $B \in \{0,1\}^b$ is a b-bit message (a *block*). The inverse (decryption) function is denoted $D_K(B')$ where B' is also a block (usually from the ciphertext). The CCM, GCM, SGCM and OCB3 are based on block ciphers, while HB2 and MASHA are not.

It is possible to identify several properties of AE schemes; we offer a non-exhaustive list. The *number of block cipher calls* used in the scheme is an important metric related to performance. A scheme is considered *online* if it is able to encrypt a message with unknown length using constant memory (this is useful, for example, if the end of the data is indicated by a null terminator or a special packet). Some schemes *only use the forward function* of the underlying block cipher (E_K), which reduces the size of software and hardware implementations. A scheme *supports preprocessing of static associated data (AD)* if the authentication of the AD depends only on the key and can be cached between different messages being sent (this is useful for a header that does not change). Some schemes are *covered by patents*, which usually discourages its use. A scheme

Table 1. Comparison of implemented AE schemes

Property	CCM	(S)GCM	OCB3	HB2	MASHA
Block cipher calls[a]	$2m + a + 2^b$	m	$m + a + 1^b$	—	—
... in key setup	0	1	1	—	—
Online	No	Yes	Yes	Yes[c]	Yes
Uses only E_K	Yes	Yes	No	—	—
Prepr. of static AD	No	Yes	Yes	No	N/A
Patent-free	Yes	Yes	No	No	No
Parallelizable	No	Yes	Yes	No	No
Standardized	Yes	(No) Yes	No	No	No
Input order	AD first	AD first	Any	AD last	N/A

[a] m, a are the number of message and AD blocks, respectively
[b] May have an additional block cipher call
[c] AD size must be fixed

is *parallelizable* if it is possible to process multiple blocks (or partially process them) in a parallel manner. Some schemes support *processing regular messages and AD in any order*, while some schemes require the processing of AD before the message, for example. The properties of the AE schemes implemented in this work are compared in Table 1.

Remarks about Security. The weak key attack against GCM, pointed out by the author of SGCM [14], has probability $n/2^{128}$ of working, where n is the number of blocks in the message; this is negligible unless the message is large. There are related key attacks against Hummingbird-2 [2,19] which, while undesirable, can be hard to apply in practice since keys are (ideally) random. Finally, there is a key-recovery attack in the multi-user setting [3] that can be applied to all schemes in this paper; however, they can be avoided by using random nonces.

4 Efficient Implementation

We have written a fast software implementation of the AE schemes in the C language, with critical functions written in assembly. The target chip was a CC430F6137 with 20 MHz clock, 32 KB flash for code and 4 KB RAM. The compiler used was the IAR Embedded Workbench version 5.30. For the AE modes based on block ciphers, we have used the AES with 128-bit keys both in software and using the AES accelerator. Our source code is available[1] to allow reproduction of our results.

The interface to the AES accelerator was written in assembly, along with a function to xor two blocks and another to increment a block.

[1] http://conradoplg.cryptoland.net/software/authenticated-encryption-for-the-msp430/

4.1 CCM

The CCM (Counter with CBC-MAC) mode [18] essentially combines the CTR mode of encryption with the CBC-MAC authentication scheme. For each message block, a counter is encrypted with the block cipher and the result xored to the message to produce the ciphertext; the counter is then incremented. The message is also xored to an "accumulator" which is then encrypted; this accumulator will become the authentication tag after all blocks are processed.

Its implementation was fairly straightforward, employing the assembly routines to xor blocks and increment the counter.

4.2 GCM

The GCM (Galois/Counter Mode) [13] employs the arithmetic of the finite field $\mathbb{F}_{2^{128}}$ for authentication and the CTR mode for encryption. For each message block, GCM encrypts the counter and xors the result into the message to produce the ciphertext; the counter is then incremented. The ciphertext is xored into an accumulator, which is then multiplied in the finite field by a key-dependent constant H. The accumulator is used to generated the authentication tag.

In order to speed up the GCM mode, polynomial multiplication was implemented in unrolled assembly with the López-Dahab (LD) [12] algorithm using 4-bit window and two lookup tables; it is described for reference in Appendix A, Algorithm 3. The first precomputation lookup table holds the product of H and all 4-bit polynomials. Each of the 16 lines of the table has 132 bits, which take 9 words. This leads to a table with 288 bytes. The additional lookup table (which can be computed from the first one, shifting each line 4 bits to the left) allows the switch from three 4-bit shifts of 256-bit blocks to a single 8-bit shift of a 256-bit block, which can be computed efficiently with the `swpb` (swap bytes) instruction of the MSP430.

4.3 SGCM

The SGCM (Sophie Germain Counter Mode) [14] is a variant of GCM that is not susceptible to weak key attacks that exist against GCM. While these attacks are of limited nature, the author claims that they should be avoided. It has the same structure as GCM, but instead of the $\mathbb{F}_{2^{128}}$ arithmetic, it uses the prime field $\mathbb{F}_p$ with $p = 2^{128} + 12451$.

Arithmetic in $\mathbb{F}_p$ can be carried out with known algorithms such as Comba multiplication. We follow the approach in [7] which takes advantage of the multiply-and-accumulate operation present in the hardware multiplier of the MSP430 family, also taking advantage of the 32-bit multiplier present in some MSP430X devices, including the CC430 series.

4.4 OCB3

The OCB3 (Offset Codebook) mode [10] also employs the $\mathbb{F}_{2^{128}}$ arithmetic (using the same reduction polynomial from GCM), but in a simplified manner: it does

not require full multiplication, but only multiplication by powers of z (the variable used in the polynomial representation of the field elements). For each i-th message block, OCB3 computes the finite field multiplication of a nonce/key-dependent constant L_0 by the polynomial z^j, where j is the number of trailing zeros in the binary representation of the block index i; the result is xored into an accumulator Δ. This accumulator is xored to the message, encrypted, and the result is xored back with Δ to generate the ciphertext. The message block is xored into another accumulator Y, which is used to generate the tag.

A lookup table with 8 entries (128 bytes) was used to hold the some precomputed values of $L_0 \cdot z^j$. Two functions were implemented in assembly: multiplication by z (using left shifts) and the function used to compute the number of trailing zeros (using right shifts).

4.5 Hummingbird-2 (HB2)

The Hummingbird-2 [5] is an authenticated encryption algorithm which is not built upon a block cipher. It processes 16-bit blocks and was specially designed for resource-constrained platforms. The small block size is achieved by maintaining an 128-bit internal state that is updated with each block processed. Authenticated data is processed after the confidential data by simply processing the blocks and discarding the ciphertext generated. The algorithm is built upon the following functions for encryption:

$$\begin{aligned} S(x) &= S_4(x[0..3]) \mid (S_3(x[4..7]) \lll 4) \\ &\quad \mid (S_2(x[8..11]) \lll 8) \mid (S_1(x[12..15]) \lll 12) \\ L(x) &= x \oplus (x \lll 6) \oplus (x \lll 10) \\ f(x) &= L(S(x)) \\ \texttt{WD16}(x, a, b, c, d) &= f(f(f(f(x \oplus a) \oplus b) \oplus c) \oplus d)\,; \end{aligned}$$

where S_1, S_2, S_3, S_4 are S-boxes and $\lll$ denotes the circular left shift of a 16-bit word. For each 16-bit message block, HB2 calls `WD16` four times, using as inputs different combinations of the message, state and key.

We have unrolled the `WD16` function. The function f is critical since it is called 16 times per block and must be very efficient; our approach is to use two precomputed lookup tables f_L, f_H each one with 256 2-byte elements, such that $f(x) = f_L[x \ \&\ \texttt{0xFF}] \oplus f_H[(x \ \&\ \texttt{0xFF00}) \gg 8]$. These tables are generated by computing $f_L[x] \leftarrow L(S_4(x[0..3]) \mid (S_3(x[4..7]) \lll 4))$ for every byte x and $f_H[x] \leftarrow L((S_2(x[8..11]) \lll 8) \mid (S_1(x[12..15]) \lll 12))$ also for every byte x. This optimization does not apply for $f^{-1}(x)$ since the inverse S-boxes are applied after the shifts in $L^{-1}(x)$. In this case, we have used precomputed lookup tables L_L, L_H such that $L(x) = L_L[x \ \&\ \texttt{0xFF}] \oplus L_H[(x \ \&\ \texttt{0xFF00}) \gg 8]$. These are computed as $f_L[x] \leftarrow L(x[0..7]), f_H[x] \leftarrow L(x[8..15] \lll 8)$ for every byte x. The four 4-bit inverse S-boxes have been merged in two 8-bit inverse S-boxes S_L^{-1}, S_H^{-1} such that $S^{-1}(x) = S_L^{-1}(x[0..7]) \mid (S_H^{-1}(x[8..15]) \lll 8)$.

4.6 MASHA

MASHA [9] is an authenticated encryption algorithm based on a stream cipher. Stream ciphers are interesting since they are often more efficient than block ciphers. However, many stream ciphers which also provide authentication either have security issues (e.g. Phelix) or performance issues. The MASHA authors propose the algorithm in order to attempt to fill this gap.

Our implementation was based on C source provided by the designers. We have changed it to reduce code size and memory footprint. The code stores the linear shift registers in circular buffers in order to avoid the actual shifts. The scheme requires multiplication, in $\mathbb{F}_{2^8}$, by four distinct constants. These are precomputed in a 256-element table which stores the multiplication of all bytes by these constants. Two such tables are required for each of the two distinct fields used by MASHA, totaling 2 KB. Since this is already large, we chose to use a byte-oriented approach for the `MixColumns` step instead of the 16-bit tailored code we will describe below. Therefore, the total space for the precomputed values becomes 2.75 KB.

4.7 Improving AES for 16-bit

We have used a software implementation of AES in order to perform comparisons with the hardware accelerator. Our implementation was based on the byte-oriented version from [6], but we have modified it to take advantage of the 16-bit platform. The first change was to improve the `AddRoundKey` function (which simply computes the xor of 128-bit blocks) in order to xor 16-bit words at a time. The second change was to improve the use of lookup tables as follows.

As it is well known, the input and output blocks of the AES can be viewed as 4×4 matrices in column-major order whose elements are in $\mathbb{F}_{2^8}$; and the AES function `SubBytes`, `ShiftRows` and `MixColumns` steps can be combined in a single one. In this step, the column j of the result matrix can be computed as

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S[a_{0,j}] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S[a_{1,j-1}] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S[a_{2,j-2}] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S[a_{3,j-3}] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix},$$

where e is the output matrix, a is the input matrix, S is the forward S-box, k is the round key matrix, and matrix indices are computed modulo four. Inspired by the 32-bit optimization of using four precomputed tables with 256 elements of with 4-byte each (totaling 4 KB), we employ the following tables:

$$T_0[a] = \begin{bmatrix} S[a] \cdot 02 \\ S[a] \end{bmatrix}, T_1[a] = \begin{bmatrix} S[a] \\ S[a] \cdot 03 \end{bmatrix}, T_2[a] = \begin{bmatrix} S[a] \cdot 03 \\ S[a] \cdot 02 \end{bmatrix}, T_3[a] = \begin{bmatrix} S[a] \\ S[a] \end{bmatrix}.$$

They consume 2 KB, half the size of the 32-bit version, providing a good compromise between the 8-bit and 32-bit oriented implementations. These tables allow the computation of column e_j as

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \end{bmatrix} = T_0[a_{0,j}] \oplus T_2[a_{1,j-1}] \oplus T_1[a_{2,j-2}] \oplus T_3[a_{3,j-3}]\,,$$
$$\begin{bmatrix} e_{2,j} \\ e_{3,j} \end{bmatrix} = T_1[a_{0,j}] \oplus T_3[a_{1,j-1}] \oplus T_0[a_{2,j-2}] \oplus T_2[a_{3,j-3}]\,.$$

4.8 Using the AES Accelerator

As previously mentioned, the AES encryption and decryption using the AES hardware accelerator requires waiting for 167 and 214 cycles, respectively, before reading the results. The key to an efficient implementation using the module is to use this "delay slot" to carry out other operations that do not depend on the result of the encryption/decryption.

For example, in the CCM mode, the counter incrementation and the xor between the message and the accumulator can be carried out while the counter is being encrypted: the counter is written to the AES accelerator, the counter is incremented, we then wait for the result of the encryption and xor the result to the message when it is ready. In CCM it is also possible to generate the ciphertext (xor the encrypted result and the message) while the accumulator is being encrypted. In the GCM mode, it is possible to increment the counter while the counter is being encrypted. In the OCB3 mode, the xor between the message and the accumulator Y can be carried out while the message, xored to Δ, is being encrypted. For reference, these computations which can be carried out in the delay slot are marked in the algorithms of Appendix A.

5 Results

The performance of the implemented AE schemes was measured for the authenticated encryption and decryption-verification of messages with 16 bytes and 4 KB, along with the Internet Performance Index (IPI) [13], which is a weighted timing for messages with 44 bytes (5%), 552 bytes (15%), 576 bytes (20%), and 1500 bytes (60%). For each message size, we have measured the time to compute all nonce-dependent values along with time for authenticated encryption and decryption-verification with 128-bit tags (except MASHA, which uses 256-bit tags). The derivation of key-dependent values is not included. For OCB3, it was assumed that the block cipher call in `init_ctr` was cached.

The timings were obtained using a development board with a CC430F6137 chip and are reported on Table 2; this data can also be viewed as throughput in Figures 1 and 2, considering a 20 MHz clock. The number of cycles taken by the algorithms was measured using the built-in cycle counter present in the CC430 models, which can be read in the IAR debugger. Stack usage was also measured using the debugger. Code size was determined from the reports produced by the compiler, adding the size for text (code) and constants.

Using the AES Accelerator. First, we analyze the results using the AES accelerator, for IPI and 4 KB messages. The GCM performance is more than 5

Table 2. Timings of implemented AE schemes for different message lengths, in cycles per byte

	Using AES accelerator			Using AES in software		
Scheme	16 bytes	IPI	4 KB	16 bytes	IPI	4 KB
Encryption						
CTR[a]	26	23	23	195	194	193
CCM	116	38	36	778	381	375
GCM	426	183	180	696	320	314
SGCM	242	89	87	567	254	250
OCB3	144	39	38	469	209	205
HB2[b]				569	200	196
MASHA[b]				3 014	182	152
Decryption						
CTR[a]	26	23	23	195	194	193
CCM	129	47	46	781	380	375
GCM	429	183	180	699	319	314
SGCM	243	89	87	571	254	250
OCB3	217	48	46	510	245	242
HB2[b]				669	297	292
MASHA[b]				3 016	182	151

[a] Non-authenticated encryption scheme included for comparison.
[b] Does not use AES.

times slower than the other schemes; this is due to the complexity of the full binary field multiplication. The SGCM is more than 50% faster than GCM, since the prime field arithmetic is much faster on this platform, specially using the 32-bit hardware multiplier. Still, it is slower than the other schemes. Both CCM and OCB3 have almost the same speed, with CCM being around 4% faster. This is surprising, since that OCB3 essentially outperforms CCM in many platforms [10]. The result is explained by the combination of two facts: the hardware support for AES, which reduces the overhead of an extra block cipher call in CCM; and the fact that the AES accelerator does not support parallelism, which prevents OCB3 from taking advantage of its support for it. We have measured that the delay slot optimization improves the encryption speed of GCM, SGCM and OCB3 by around 12% and CCM by around 24%.

Using the AES in Software. We now consider the performance using the software AES implementation, for large messages. For reference, the block cipher takes 180 cycles per byte to encrypt and 216 cycles per byte to decrypt. The CCM mode becomes slower due to the larger overhead of the extra block cipher call. The GCM is still slower than OCB3 due to its expensive field multiplication. The SGCM is also faster than GCM, but the improvement is diluted to 20% with the software AES. The MASHA cipher is the fastest, followed by Hummingbird-2,

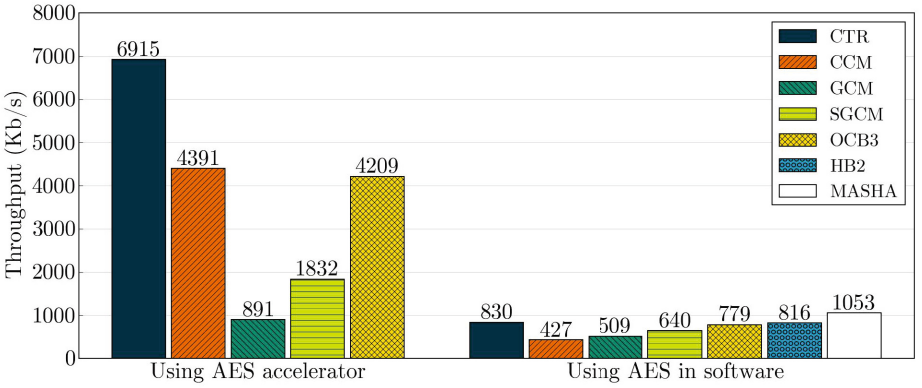

Fig. 1. Encryption throughput in Kbps of CTR and AE schemes for 4 KB messages at 20 MHz

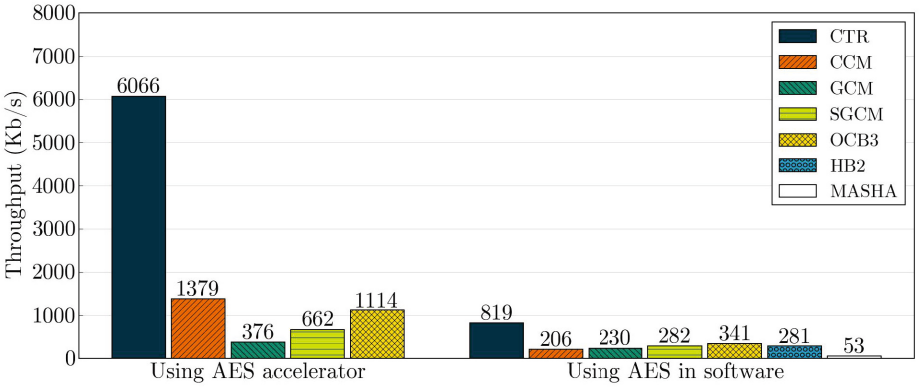

Fig. 2. Encryption throughput in Kbps of CTR and AE schemes for 16-byte messages at 20 MHz

which is 22% slower. Interestingly, Hummingbird-2 fails to outperform AES in CTR mode, which is surprising since it is specially tailored for the platform (of course, it must be considered that it provides authentication while AES-CTR by itself does not).

AES Accelerator vs. AES in Software. Using the AES accelerator, it is possible to encrypt in the CTR mode approximately 8 times faster than using AES in software; and it is possible to encrypt with CCM approximately 10 times faster for encryption and 8 times faster for decryption. The AES accelerator speedup for GCM, SGCM and OCB3 is smaller (around 1.7, 2.8, and 5.4, respectively), due to the larger software overhead.

Encryption vs. Decryption. When considering the usage of the AES accelerator, GCM has roughly the same performance in encryption and decryption, since

the algorithm for both is almost equal; the same applies for SGCM. For both CCM and OCB3, decryption is around 25% and 20% slower, respectively. This is explained by the differences in the data dependencies of the decryption, which prevents the useful use of the delay slot, and that D_K (used by OCB3) is slower than E_K in the AES accelerator. Considering now the usage of the AES in software, encryption and decryption have the same performance in CCM and GCM (since there is no delay slot now) as well as in MASHA. However, decryption is almost 18% slower for OCB3, since the underlying block cipher decryption is also slower than the encryption. The decryption in Hummingbird-2 is almost 50% slower due to the $f^{-1}(x)$ function not being able to be fully precomputed, in contrast to $f(x)$. It is interesting to note that the decryption timings are often omitted in the literature, even though they may be substantially different from the encryption timings.

Performance for Small Messages. The timings for 16-byte messages are usually dominated by the computation of nonce-dependent values. The CCM using software AES has the second worst performance since all of its initialization is nonce-dependent (almost nothing is exclusively key-dependent) and it includes two block cipher calls. When using the AES accelerator, this overhead mostly vanishes, and CCM becomes the faster scheme. The nonce setup of GCM is very cheap (just a padding of the nonce) while the nonce setup of OCB3 requires the left shift of an 192-bit block by 0–63 bits. Still, the GCM performance for 16-byte messages is worse than OCB3 since it is still dominated by the block processing. Hummingbird-2 loses to OCB3 due to its larger nonce setup and tag generation. The greatest surprise is the MASHA performance which is almost four times slower than CCM, making it the slowest scheme for small messages. This result is explained by the fact that its nonce setup and tag generation are very expensive, requiring more than 20 state updates each (which take roughly the same time as encrypting ten 128-bit blocks).

Further Analysis. In order to evaluate our AES software implementation, consider the timings from [4] (also based on [6]) which achieved 286 Kbps at 8 MHz in the ECB mode. Scaling this to 20 MHz we get 716 Kbps, while our ECB implementation achieved 889 Kbps. We conclude that our 16-bit implementation is 24% faster then the byte-oriented implementation.

Table 3 lists the ROM and RAM usage for programs implementing AE schemes for both encryption and decryption, using the AES accelerator. The reported sizes refer only to the code related to the algorithms and excludes the benchmark code. We recall that the MSP430X model we have used features 32 KB of flash for code and 4 KB RAM. The code for GCM is large due to the unrolled $\mathbb{F}_{2^{128}}$ multiplier, while the code for CCM is the smallest since it mostly relies on the block cipher. The RAM usage follows the same pattern: GCM has the second largest usage, since it has the largest precomputation table; the Hummingbird-2 cipher (followed by CCM) has the smallest RAM usage since it requires no runtime precomputation at all. The MASHA cipher requires the largest code space, due to the many precomputed tables used; this can be

reduced by sacrificing speed. When using the software AES implementation, 2 904 additional ROM bytes are required for CCM, GCM and SGCM (which use E_K only) and 5 860 additional ROM bytes are required for OCB3.

Table 3. ROM and RAM (stack) usage of AE schemes, in bytes. When using software AES, 2 904 additional ROM bytes are required for CCM, GCM and SGCM and 5 860 bytes for OCB3.

	CTR	CCM	GCM	SGCM	OCB3	HB2	MASHA
ROM	130	1 094	4 680	2 172	1 724	3 674	5 602
RAM	100	258	886	322	538	196	499

5.1 Related Work

A commercial 128-bit AES implementation for the MSP430 [8] achieves 340 cycles per byte for encryption and 550 cpb for decryption, in ECB mode, using 2536 bytes. Our implementation provides 180 cpb and 216 cpb, respectively, but uses 5860 bytes. With space-time tradeoffs, it should be feasible to achieve similar results, but we have not explored them.

Simplicio Jr. et al. [16] have implemented EAX, GCM, LetterSoup, OCB2 and CCFB+H for the MSP430, using Curupira as the underlying block cipher. The EAX mode is [1] is described as an "cleaned-up" CCM and has similar performance. The authors report the results in milliseconds, but do not state the clock used. Assuming a 8 MHz clock, their timings (in cycles per byte, considering their timings for 60-byte messages and our timings for 16-byte messages) are 1 733 cpb for EAX, 5 133 cpb for GCM, 1 680 cpb for LetterSoup, 1 506 cpb for OCB2 and 2 266 cpb for CCFB+H with 8-byte tag. Our CCM is 2.2 times faster than their EAX, while our GCM is 7.3 times faster, and our OCB3 3.2 times faster than their OCB2. This difference can probably be explained by the fact that the authors have not optimized the algorithms for performance.

In [4], the encryption performance using the AES module present in the CC2420 transceiver is studied, achieving 110 cycles per byte. This is still 5 times slower than our results for the CTR mode, probably because the CC2420 is a peripheral and communicating with it is more expensive.

The Dragon-MAC [11] is based on the Dragon stream cipher. Its authors describe an implementation for the MSP430 that achieves 21.4 cycles per byte for authenticated encryption (applying Dragon then Dragon-MAC), which is faster than all timings in this work. However, it requires 18.9 KB of code. Our CCM implementation using the AES accelerator is 1.7 times slower, but 11 times smaller; while our HB2 is 9.2 times slower and 5.1 times smaller.

The Hummingbird-2 timings reported for the MSP430 in its paper [5] are about 6% and 2% faster for encryption and decryption than the timings we have obtained. However, the authors do not describe their optimization techniques,

nor the exact MSP430 model used and their timing methodology, making it difficult to explain their achieved speed. However, we believe that our implementation is good enough for comparisons. Furthermore, by completely unrolling the encryption and decryption functions, we were able to achieve timings 3% and 4% faster than theirs, increasing code size by 296 and 432 bytes, respectively.

6 Conclusion and Future Work

The CCM and OCB3 modes were found to provide similar speed results using the AES accelerator, with CCM being around 5% faster. While OCB3 is the fastest scheme in many platforms, we expect CCM to be faster whenever a non-parallel AES accelerator is available. This is the case for the MSP430X models studied and is also the case for other platforms, for example, the AVR XMEGA microcontroller with has an analogue AES module.

The CCM appears to be the best choice for MSP430X models with AES accelerator considering that it also consumes less code space and less stack RAM. If one of the undesirable properties of CCM must be avoided (not being online, lack of support for preprocessing of static AD), a good alternative is the EAX mode [1] and should have performance similar to CCM. Considering software-only schemes, it is harder to give a clear recommendation: SGCM, OCB3 and HB2 provide good results, with distinct advantages and downsides. The GCM mode, even though it has many good properties, does not appear to be adequate in software implementation for resource-constrained platforms since it requires very large lookup tables in order to be competitive.

Some other relevant facts we have found are that Hummingbird-2 is slower than AES; that SGCM is 50% faster than GCM when using the AES accelerator and 20% when not; and that OCB3 and Hummingbird-2 in particular have a decryption performance remarkably slower than encryption (18% and 50% respectively). MASHA has great speed for large enough messages (29% faster than the second fastest, HB2) but very low performance for small messages (almost 4 times slower than the second slowest, CCM). For this reason, we believe there is still the need for a fast, secure and lightweight authenticated encryption scheme based on a stream cipher.

For future works it would be interesting to implement and compare lightweight encrypt-and-authenticate or authenticated encryption schemes such as LetterSoup [15] and Rabbit-MAC [17] for the MSP430X. Another possible venue for research is to study the efficient implementation of authenticated encryption using the AES accelerator featured in other platforms such as the AVR XMEGA and devices based on the ARM Cortex such as the EFM32 Gecko, STM32 and LPC1800.

References

1. Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 389–407. Springer, Heidelberg (2004)

2. Chai, Q., Gong, G.: A cryptanalysis of HummingBird-2: The differential sequence analysis. Cryptology ePrint Archive, Report 2012/233 (2012), http://eprint.iacr.org/
3. Chatterjee, S., Menezes, A., Sarkar, P.: Another Look at Tightness. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 293–319. Springer, Heidelberg (2012)
4. Didla, S., Ault, A., Bagchi, S.: Optimizing AES for embedded devices and wireless sensor networks. In: Proceedings of the 4th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, pp. 4:1–4:10 (2008)
5. Engels, D., Saarinen, M.-J.O., Schweitzer, P., Smith, E.M.: The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 19–31. Springer, Heidelberg (2012)
6. Gladman, B.: AES and combined encryption/authentication modes (2008), http://gladman.plushost.co.uk/oldsite/AES/
7. Gouvêa, C.P.L., López, J.: Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. Journal of Cryptographic Engineering 2(1), 19–29 (2012)
8. Institute for Applied Information Processing and Communication: Crypto software for microcontrollers - Texas Instruments MSP430 microcontrollers (2012), http://jce.iaik.tugraz.at/sic/Products/Crypto_Software_for_Microcontrollers/Texas_Instruments_MSP430_Microcontrollers
9. Kiyomoto, S., Henricksen, M., Yap, W.-S., Nakano, Y., Fukushima, K.: MASHA – Low Cost Authentication with a New Stream Cipher. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 63–78. Springer, Heidelberg (2011)
10. 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)
11. Lim, S.Y., Pu, C.C., Lim, H.T., Lee, H.J.: Dragon-MAC: Securing wireless sensor networks with authenticated encryption. Cryptology ePrint Archive, Report 2007/204 (2007), http://eprint.iacr.org/
12. López, J., Dahab, R.: High-Speed Software Multiplication in $\mathbb{F}_{2^m}$. In: Roy, B., Okamoto, E. (eds.) INDOCRYPT 2000. LNCS, vol. 1977, pp. 203–212. Springer, Heidelberg (2000)
13. McGrew, D.A., Viega, J.: The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004)
14. Saarinen, M.J.O.: SGCM: The Sophie Germain counter mode. Cryptology ePrint Archive, Report 2011/326 (2011), http://eprint.iacr.org/
15. Simplicio Jr., M.A., Barbuda, P.F.F.S., Barreto, P.S.L.M., Carvalho, T.C.M.B., Margi, C.B.: The MARVIN message authentication code and the LETTERSOUP authenticated encryption scheme. Security and Communication Networks 2(2), 165–180 (2009)
16. Simplicio Jr., M.A., de Oliveira, B.T., Barreto, P.S.L.M., Margi, C.B., Carvalho, T.C.M.B., Naslund, M.: Comparison of authenticated-encryption schemes in wireless sensor networks. In: 2011 IEEE 36th Conference on Local Computer Networks (LCN), pp. 450–457 (2011)

17. Tahir, R., Javed, M., Cheema, A.: Rabbit-MAC: Lightweight authenticated encryption in wireless sensor networks. In: International Conference on Information and Automation, ICIA 2008, pp. 573–577 (2008)
18. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM) (2002), http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html
19. Zhang, K., Ding, L., Guan, J.: Cryptanalysis of Hummingbird-2. Cryptology ePrint Archive, Report 2012/207 (2012), http://eprint.iacr.org/

A Algorithms

Algorithm 1 presents CCM, where the function `format` computes a header block B_0 (which encodes the tag length, message length and nonce), the blocks $A_1, \ldots, A_a$ (which encode the length of the associated data along with the data itself) and the blocks $M_1, \ldots, M_m$ which represent the original message. The function `init_ctr` returns the initial counter based on the nonce. The function `inc` increments the counter.

Algorithm 1. CCM encryption

Input: Message M, additional data A, nonce N, key K
Output: Ciphertext C, authentication tag T with t bits

1. $B_0, A_1, \ldots, A_a, M_1, \ldots, M_m \leftarrow \texttt{format}(N, A, M)$
2. $Y \leftarrow E_K(B_0)$
3. **for** $i \leftarrow 1$ **to** a **do**
4. $\quad Y \leftarrow E_K(A_i \oplus Y)$
5. **end for**
6. $J \leftarrow \texttt{init\_ctr}(N)$
7. $S_0 \leftarrow E_K(J)$
8. $J \leftarrow \texttt{inc}(J)$
9. **for** $i \leftarrow 1$ **to** m **do**
10. $\quad U \leftarrow E_K(J)$
11. $\quad J \leftarrow \texttt{inc}(J)$ {delay slot}
12. $\quad S \leftarrow M_i \oplus Y$ {delay slot}
13. $\quad Y \leftarrow E_K(S)$
14. $\quad C_i \leftarrow M_i \oplus U$ {delay slot}
15. **end for**
16. $T \leftarrow Y[0..t-1] \oplus S_0[0..t-1]$

Algorithm 2 describes GCM, where the function `init_ctr` initializes the counter and the function `inc_ctr` increments the counter. The operation $A \cdot B$ denotes the multiplication of A and B in $\mathbb{F}_{2^{128}}$. The mode benefits from precomputed lookup tables since the second operand is fixed for all multiplications (lines 6, 15 and 18 from Algorithm 1). The LD multiplication with two tables, used in the field multiplication, is described in Algorithm 3.

OCB3 is described in Algorithm 4, where the function `init_delta` derives a value from the nonce and it may require a block cipher call, as explained

Algorithm 2. GCM encryption

Input: Message M, additional data A, nonce N, key K
Output: Ciphertext C, authentication tag T with t bits

1. $A_1, \ldots, A_a \leftarrow A$
2. $M_1, \ldots, M_m \leftarrow M$
3. $H \leftarrow E_K(0^{128})$
4. $Y \leftarrow 0^{128}$
5. **for** $i \leftarrow 1$ **to** a **do**
6. $\quad Y \leftarrow (A_i \oplus Y) \cdot H$
7. **end for**
8. $J \leftarrow \texttt{init\_ctr}(N)$
9. $S_0 \leftarrow E_K(J)$
10. $J \leftarrow \texttt{inc}(J)$
11. **for** $i \leftarrow 1$ **to** m **do**
12. $\quad U \leftarrow E_K(J)$
13. $\quad J \leftarrow \texttt{inc}(J)$ {delay slot}
14. $\quad C_i \leftarrow M_i \oplus U$
15. $\quad Y \leftarrow (C_i \oplus Y) \cdot H$
16. **end for**
17. $L \leftarrow [len(A)]_{64} \,||\, [len(M)]_{64}$
18. $S \leftarrow (L \oplus Y) \cdot H$
19. $T \leftarrow (S \oplus S_0)[0..t-1]$

Algorithm 3. López-Dahab multiplication in $\mathbb{F}_{2^{128}}$ for 16-bit words and 4-bit window, using 2 lookup tables

Input: $a(z) = a[0..7], b(z) = b[0..7]$
Output: $c(z) = c[0..15]$

1. Compute $T_0(u) = u(z)b(z)$ for all polynomials $u(z)$ of degree lower than 4.
2. Compute $T_1(u) = u(z)b(z)z^4$ for all polynomials $u(z)$ of degree lower than 4.
3. $c[0..15] \leftarrow 0$
4. **for** $k \leftarrow 1$ **down to** 0 **do**
5. $\quad$ **for** $i \leftarrow 0$ **to** 7 **do**
6. $\quad\quad u_0 \leftarrow (a[i] \gg (8k)) \bmod 2^4$
7. $\quad\quad u_1 \leftarrow (a[i] \gg (8k+4)) \bmod 2^4$
8. $\quad\quad$ **for** $j \leftarrow 0$ **to** 8 **do**
9. $\quad\quad\quad c[i+j] \leftarrow c[i+j] \oplus T_0(u_0)[j] \oplus T_1(u_1)[j]$
10. $\quad\quad$ **end for**
11. $\quad$ **end for**
12. $\quad$ **if** $k > 0$ **then**
13. $\quad\quad c(z) \leftarrow c(z)z^8$
14. $\quad$ **end if**
15. **end for**
16. **return** c

later. The function $\texttt{ntz}(i)$ returns the number of trailing zeros in the binary representation of i (e.g. $\texttt{ntz}(1) = 0$, $\texttt{ntz}(2) = 1$). The function $\texttt{getL}(L_0, x)$ computes the field element $L_0 \cdot z^x$ and can benefit from a precomputed lookup table. Notice that the multiplication by z is simply a left shift of the operand by one bit, discarding the last bit and xoring the last byte of the result with 135 (which is the representation of $z^7 + z^2 + z^1 + 1$) if the discarded bit was 1. The function `hash` authenticates the additional data and is omitted for brevity.

Algorithm 4. OCB3 mode encryption

Input: Message M, additional data A, nonce N, key K
Output: Ciphertext C, authentication tag T with t bits

1. $A_1, \ldots, A_a \leftarrow A$
2. $M_1, \ldots, M_m \leftarrow M$
3. $L_* \leftarrow E_K(0^{128})$
4. $L_\$ \leftarrow L_* \cdot z$
5. $L_0 \leftarrow L_\$ \cdot z$
6. $Y \leftarrow 0^{128}$
7. $\Delta \leftarrow \texttt{init\_delta}(N, K)$
8. **for** $i \leftarrow 1$ **to** m **do**
9. $\quad \Delta \leftarrow \Delta \oplus \texttt{getL}(L_0, \texttt{ntz}(i))$
10. $\quad U \leftarrow E_K(M_i \oplus \Delta)$
11. $\quad Y \leftarrow Y \oplus M_i$ {delay slot}
12. $\quad C_i \leftarrow U \oplus \Delta$
13. **end for**
14. $\Delta \leftarrow \Delta \oplus L_\$$
15. $F \leftarrow E_K(Y \oplus \Delta)$
16. $G \leftarrow \texttt{hash}(K, A)$
17. $T \leftarrow (F \oplus G)[0..t-1]$

Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output

Benedikt Gierlichs[1], Jörn-Marc Schmidt[2], and Michael Tunstall[3]

[1] KU Leuven Dept. Electrical Engineering-ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
benedikt.gierlichs@esat.kuleuven.be
[2] Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
joern-marc.schmidt@iaik.tugraz.at
[3] Department of Computer Science, University of Bristol
Merchant Venturers Building, Woodland Road
Bristol BS8 1UB, United Kingdom
tunstall@cs.bris.ac.uk

Abstract. Implementation attacks pose a serious threat for the security of cryptographic devices and there are a multitude of countermeasures that are used to prevent them. Two countermeasures used in implementations of block ciphers to increase the complexity of such attacks are the use of dummy rounds and redundant computation with consistency checks to prevent fault attacks. In this paper we present several countermeasures based on the idea of infective computation. Our countermeasures ensure that a fault injected into a cipher, dummy, or redundant round will infect the ciphertext such that an attacker cannot derive any information on the secret key being used. This has one clear advantage: the propagation of faults prevents an attacker from being able to conduct any fault analysis on any corrupted ciphertexts. As a consequence, there is no need for any test at the end of an implementation to determine if a fault has been injected and a ciphertext can always be returned.

Keywords: Implementation Attacks, Dummy Rounds, Infective Computation.

1 Introduction

Implementation attacks are currently one of the most powerful threats for cryptographic devices. Instead of considering a cryptographic device like a smart card as a black box, these attacks try to benefit from characteristics of the implementation, either by measuring properties of the device during a computation or by actively manipulating the execution of an algorithm. These can be grouped into two main types: Passive attacks based on measuring and analyzing properties

A. Hevia and G. Neven (Eds.): LATINCRYPT 2012, LNCS 7533, pp. 305–321, 2012.

of the device (referred to as a side-channel attack), and active attacks where an attacker seeks to modify the behavior of a device (referred to as a fault attack). Both attack methods may reveal cryptographic keys used inside a device, irrespective of the theoretical (black-box) security of the underlying cryptographic algorithm.

Side-channel analysis was introduced to the cryptographic community by Kocher [1], who noted that the time taken to compute a cryptographic algorithm can potentially reveal the cryptographic key being used. The idea of the attack was later extended by Kocher et al. [2], who noted that the power consumption of a microprocessor depends on the code being executed. Moreover, the instantaneous power consumption is dependent on the data being manipulated at that point in time (typically proportional to the Hamming weight of the data [3]). This allows information on cryptographic keys to be determined, since one can verify a hypothetical set of values that occur after the input is combined with a key. It was later observed that the same attacks could be applied to the electromagnetic emanations surrounding a microprocessor [4,5]. A suitable countermeasure to such attacks is to mask all intermediate states of an algorithm with some random value [6,7]. One can further complicate the task of an attacker by executing an algorithm in some non-deterministic order, such that data is not processed at a set point in time. This spreads the information an attacker would want to exploit over numerous points in time. One way to introduce time randomization in an implementation of a block cipher is the use of dummy rounds. That is, each round observed by an attacker will be a dummy round with some probability, and produce false hypotheses for an attacker.

Fault analysis seeks to exploit the effect of a fault inserted into an instance of a cryptographic algorithm [8,9,10]. When implementing a block cipher a simple countermeasure is, as for side-channel analysis, to use time randomization, e.g. dummy rounds, that can render precise fault injection more difficult. However, dummy rounds do not affect an attacker directly, since injecting a fault into a dummy round will have no observable effect. Another simple countermeasure is to introduce redundancy in a certain number of rounds at the beginning and the end of a block cipher. This can, for instance, involve repeating the execution of functions in software implementations or using parallel blocks in hardware. At the end of the redundant computation the implementation checks if all redundantly computed results are equal and suppresses the output if they are not.

Contribution. In this paper we show how one can link cipher rounds, dummy rounds, and redundant rounds to further increase the security in implementations of block ciphers. The countermeasures disturb the ciphering process each time a fault is injected into a cipher, dummy, or redundant round, and give an attacker no information with which to deduce information on a secret key. As a consequence, an implementation does not need to use any consistency checks, which may themselves become targets for a fault attack and which inherently leak information about the success of fault injection.

Our countermeasures even provide some protection against an adversary who attempts to inject the *same* fault in both (all) branches of redundant computation to bypass the final consistency check. While the check actually aids the adversary since it will only output the ciphertext if the fault injection was successful, our proposed algorithms always output a ciphertext (exploitable or not). Thus, an adversary has to analyze the output without knowing whether the fault injection was successful and the output contains exploitable information, or not.

Organization. The rest of this paper is organized as follows: Section 2 provides background information on fault analysis of block cipher implementations and countermeasures. Section 3 presents our countermeasures and their application to S-P networks and Feistel ciphers. Section 4 provides the security evaluation of the countermeasures in both cases. Section 5 explains how the countermeasures can be hardened to prevent the detection of dummy rounds by side-channel analysis and Section 6 concludes the paper.

2 Background

Since the introduction of fault attacks by Boneh et al. [9], the idea of exploiting erroneous results to reveal secrets was applied to many cryptographic algorithms. The ideas are steadily improved for reducing the number of faults required. For example, it is noted in [11] that a *single* fault may be sufficient to break an AES implementation by using differential fault analysis. For such attacks, the same plaintext is encrypted twice, while a fault is injected during one of the computations. From the resulting difference in the outputs, the secret key or parts of it are derived. In contrast to differential fault analysis, collision fault analysis [8] relies on finding a plaintext that maps to the *same* output as a faulty encryption of a different plaintext. This technique is often applied to attack early rounds of an algorithm. Safe-error attacks [12] and ineffective fault analysis [10] do not require the actual output of the computation. The information whether the result is erroneous is sufficient. The same idea can also be used to easily detect dummy operations in a computation.

In order to detect fault attacks some kind of redundancy is typically introduced into implementations of cryptographic algorithms. A straightforward approach is computing the same algorithm twice and comparing the results. If the results differ, the output of the algorithm is suppressed. In order to protect the algorithm without repeating it, the proposals include limiting the repetition to a few rounds, introducing a parity byte for the state [13], and generating digest values that are tailored to the operations of the cipher [14]. While these approaches try to reduce the overhead compared to doubling the cipher, other proposals aim at a higher detection rate for injected faults, such as involving the inverse round-function of the cipher for the check [15] and enlarging the field the algorithm computes in [16].

However, irrespective of how the redundancy is introduced, the check of the result before it is released is a potential target for an adversary. Kim and Quisquater demonstrated that one can attack a cryptographic algorithm and

then inject a fault in the verification stage [17]. The natural response would be to make the verification itself redundant but van Woudenberg et al. [18] have shown that three faults can be used to attack two tests after the execution of a cryptographic algorithm, i.e. inject one fault in the algorithm itself and use two faults to overcome the redundant verification.

In order to prevent an adversary from using multiple fault-injection to bypass checking routines, Yen and Joye introduced the principle of infective computing [12]. Their proposal defines a method of generating RSA signatures where any fault injected into a computation changes the output of the cryptographic algorithm in such a way that it does not reveal any secret information. That is, the output of the cryptographic algorithm should not be exploitable. The usual way to implement infective computation is to introduce a (secret) error in an input, then to compute the result, and finally to remove the effect of the previously introduced error. If a fault is injected in the computation, then the output of the algorithm is incorrect and cannot be exploited because the initial error is unknown. An alternative way is to introduce additional computation on secret data in the algorithm that will have no effect on the result, if no fault is injected.

In the following section we describe how the latter approach can be applied to block ciphers using dummy rounds.

3 Smart Use of Dummy Rounds and Redundant Computation

We first explain how our approach can be used to make smart use of dummy rounds. This leads to an algorithm where dummy rounds can no longer be identified by fault injection, and where a fault injected into a dummy round renders the ciphertext useless to an attacker. Then we extend the approach to redundant implementations with dummy rounds.

We provide algorithms for the application of our countermeasures to S-P networks and Feistel ciphers that use the following notation:

`BlockCipher` — The entire block cipher under consideration which takes a plaintext P and enciphers it with a secret key K to produce a ciphertext C. We will consider a block cipher that consists of n rounds.

`RoundFunction` — The round function of the block cipher. It operates on a given register and requires the correct subkey k_i for that round.

`RandomBit` — This function returns a random bit that governs whether a dummy round will occur or not. If this bit is equal to zero a dummy round will be computed, otherwise a cipher round will take place. This function can be replaced by any suitable function that returns one bit, if, for example, one wants to limit the number of dummy rounds that could occur in a given instantiation of the block cipher.

We denote a bitwise logical AND operation by $\wedge$, a bitwise logical NOT operation by $\neg$, and a bitwise logical exclusive-OR (XOR) operation by $\oplus$.

3.1 Smart Use of Dummy Rounds

We propose a countermeasure where a dummy round takes a secret input value β that is known to produce a result of β after one round when combined with the secret dummy round-key k_0. That is

$$\texttt{RoundFunction}(\beta, k_0) = \beta\,.$$

The result of the dummy round is then XORed into the cipher state, and then XORed with β, ensuring that any fault is propagated into the block cipher. The result of the dummy round further overwrites β held in registers. This ensures that any fault is propagated into subsequent dummy rounds. The same operations are conducted when a cipher round is computed. This means that any fault that modifies the output of a dummy round will affect the input of *every* subsequent round, making any collision or differential fault analysis impossible.

Application to S-P Networks. Algorithm 1 shows how the countermeasure can be applied to a straightforward implementation of a S-P network. The

Algorithm 1. S-P network with smart dummy rounds

Input: P, k_i for $i \in \{1, \ldots, n+1\}$ ($n+1$ subkeys from key K), (β, k_0).
Output: $C = \texttt{BlockCipher}(P, K)$

State: $R_0 \leftarrow P$; Dummy state $R_1 \leftarrow \beta$; $i \leftarrow 1$;
while $i \leq n$ **do**
  $\lambda \leftarrow \texttt{RandomBit}()$; // $\lambda = 0$ implies a dummy round
  $\kappa \leftarrow i\,\lambda$;
  $R_{\neg\lambda} \leftarrow \texttt{RoundFunction}(R_{\neg\lambda}, k_\kappa)$ // infection of the dummy state
  $R_0 \leftarrow R_0 \oplus R_1 \oplus \beta$; // infection of the cipher state
  $i \leftarrow i + \lambda$;
end
return R_0

algorithm clearly achieves the goal of propagating a fault injected into a dummy round into *every* subsequent round.

Application to Feistel Ciphers. Another commonly used mechanism for block ciphers is the Feistel structure. This operates by dividing the input of one round into two equally sized sets and using a round function on one set before combining it with the other set using an XOR. For example, if we define L_i and R_i as the left and right hand inputs to the i^{th} round of a block cipher respectively, then L_{i+1} and R_{i+1} are computed in the following manner:

$$\begin{aligned} L_{i+1} &\leftarrow R_i \\ R_{i+1} &\leftarrow \texttt{RoundFunction}(R_i, k_{i+1}) \oplus L_i. \end{aligned}$$

Algorithm 2 shows how the countermeasure can be applied to a straightforward implementation of a Feistel cipher. It uses the following additional notation: let P_r be the right-hand side and P_l be the left-hand side of the plaintext, respectively. Further, let α be some arbitrary, constant value where the relationship between α, β and k_0 is $\texttt{RoundFunction}(\beta, k_0) \oplus \alpha = \beta$.

Algorithm 2. Feistel cipher with smart dummy rounds

Input: P, k_i for $i \in \{1, \ldots, n\}$ (n subkeys from key K), (β, α, k_0).
Output: $C = \texttt{BlockCipher}(P, K)$

State $R_0 \leftarrow P_r$; Dummy state $R_1 \leftarrow \beta$; $i \leftarrow 1$;
State $T_0 \leftarrow P_l$; $T_1 \leftarrow \alpha$;
while $i \leq n$ **do**
  $\lambda \leftarrow \texttt{RandomBit}()$; // $\lambda = 0$ implies a dummy round
  $\kappa \leftarrow i\,\lambda$;
  $R_{\neg\lambda} \leftarrow \texttt{RoundFunction}(R_{\neg\lambda}, k_\kappa) \oplus T_{\neg\lambda}$ // infection of the dummy state
  $R_0 \leftarrow R_0 \oplus R_1 \oplus \beta$; // infection of the cipher state
  $T_0 \leftarrow T_0 \oplus R_1 \oplus \beta$; // infection of the cipher state
  $i \leftarrow i + \lambda$;
end
return $T_0 \| R_0$

We can note that a fault in a dummy round will affect both R_0 and T_0 and therefore provoke a larger change in the resulting ciphertext.

3.2 Smart Use of Redundant Rounds and Dummy Rounds

Algorithms 1 and 2 both achieve the goal of propagating faults injected into a *dummy* round into *every* subsequent round. However, both algorithms still require the output of some of the first and last rounds to be checked to prevent collision and differential fault analysis based on a fault injected into a *cipher* round.

To overcome this limitation we further propose algorithms where each round is repeated and any fault in a single round (cipher, dummy, or redundant round) will affect the resulting ciphertext such that no information is available to an attacker. This means that a verification stage is not necessary since an attacker will not receive any information from a fault. As before, an attacker will not be able to determine if any particular round is a dummy round by means of fault injection.

Note that repeating each round implies that each round can be observed twice through some side-channel, which can ease side-channel analysis. However, the time randomization due to the dummy rounds provides some level of protection. In addition, although each cipher round is repeated, this will occur in a somewhat random manner. That is, a cipher round may be followed by a dummy round or a redundant round with some probability determined by how `RandomBit` is

defined, and the round counter only increases once the redundant round has been computed.

As above, we assume that β and k_0 are chosen such that the result of the round function is β. That is

$$\texttt{RoundFunction}(\beta, k_0) = \beta\,.$$

The algorithms also use additional notation:

`SNLF` — This stands for Some NonLinear Function, which is used to ensure that any fault in the block cipher will not provide an attacker with any information. We discuss the reasons for this in more detail in Section 4. In our algorithms we assume that $\texttt{SNLF}(0) \mapsto 0$.

Application to S-P Networks. Algorithm 3 shows how the countermeasure can be applied to a redundant implementation of an S-P network.

Algorithm 3. Redundant S-P Network with Dummy Rounds

Input: P, k_i for $i \in \{1, \ldots, n+1\}$ ($n+1$ subkeys from key K), (β, k_0).
Output: $C = \texttt{BlockCipher}(P, K)$

State $R_0 \leftarrow P$; Redundant state $R_1 \leftarrow P$; Dummy state $R_2 \leftarrow \beta$;
$C_0 \leftarrow 0$; $C_1 \leftarrow 0$; $C_2 \leftarrow \beta$; $i \leftarrow 0$;
while $i < 2\,n$ **do**
  $\lambda \leftarrow \texttt{RandomBit}()$; // $\lambda = 0$ implies a dummy round
  $\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda)$;
  $\zeta \leftarrow \lceil i/2 \rceil\, \lambda$; // ζ is actual round counter, 0 for dummy
  $R_\kappa \leftarrow \texttt{RoundFunction}(R_\kappa, k_\zeta)$;
  $C_\kappa \leftarrow R_\kappa \oplus C_2 \oplus \beta$; // infect C_κ to propagate a fault
  $\epsilon \leftarrow \lambda(\neg(i \wedge 1)) \cdot \texttt{SNLF}(C_0 \oplus C_1)$; // check if i is even
  $R_2 \leftarrow R_2 \oplus \epsilon$;
  $R_0 \leftarrow R_0 \oplus \epsilon$;
  $i \leftarrow i + \lambda$;
end
$R_0 \leftarrow R_0 \oplus \texttt{RoundFunction}(R_2, k_0)$;
return R_0

Algorithm 3 progresses by computing the same round twice before advancing to the next round using i as a counter. When i is an even number the difference between the result of a cipher round and a redundant round should be equal to zero and any difference is XORed into β. This difference also goes through some nonlinear function to make it difficult for an attacker to make any hypotheses about any fault that has been induced. If i is an odd number, or a dummy round occurs, the difference between the cipher round and the redundant round

is multiplied by zero since it will be non-zero during the normal functioning of the algorithm. Tables 1(a), 1(b) and 1(c) illustrate the functioning of the algorithm with examples.

Table 1. Examples of Algorithm 3

(a) A dummy round. A fault will change C_2 infecting every subsequent round.

$\lambda = 0$
$\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda) = 2$
$\zeta \leftarrow \lceil i/2 \rceil\ \lambda = 0$
$R_2 \leftarrow \texttt{RoundFunction}(R_2, k_0) = \beta$
$C_2 \leftarrow R_2 \oplus C_2 \oplus \beta = \beta$
$\epsilon \leftarrow 0 \cdot \texttt{SNLF}(C_0 \oplus C_1) = 0$
$R_2 \leftarrow R_2 \oplus \epsilon$
$R_0 \leftarrow R_0 \oplus \epsilon$
$i \leftarrow i + 0$

(b) A round where i is even. A fault will change R_0 that will infect every subsequent round where i is odd and every round after the next dummy round.

$i = \text{even}, \lambda = 1$
$\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda) = 0$
$\zeta \leftarrow \lceil i/2 \rceil$
$R_0 \leftarrow \texttt{RoundFunction}(R_0, k_\zeta)$
$C_0 \leftarrow R_0 \oplus C_2 \oplus \beta = R_0$
$\epsilon \leftarrow 0 \cdot \texttt{SNLF}(C_0 \oplus C_1) = 0$
$R_2 \leftarrow R_2 \oplus \epsilon$
$R_0 \leftarrow R_0 \oplus \epsilon$
$i \leftarrow i + 1$

(c) A round where i is odd. A fault will change R_1 that will infect every subsequent round where i is even and every round after the next dummy round.

$i = \text{odd}, \lambda = 1$
$\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda) = 1$
$\zeta \leftarrow \lceil i/2 \rceil$
$R_1 \leftarrow \texttt{RoundFunction}(R_1, k_\zeta)$
$C_1 \leftarrow R_1 \oplus C_2 \oplus \beta = R_1$
$\epsilon \leftarrow 1 \cdot \texttt{SNLF}(C_0 \oplus C_1)$
$R_2 \leftarrow R_2 \oplus \epsilon$
$R_0 \leftarrow R_0 \oplus \epsilon$
$i \leftarrow i + 1$

An extra dummy round is added to the end of the algorithm to ensure that any fault that has been propagated through the algorithm masks the ciphertext in such a way that no information is available to an attacker. Moreover, this will ensure the countermeasure is effective if an attacker is able to affect the `RandomBit` function such that no dummy rounds occur.

The extra dummy round and the use of the nonlinear function are discussed in more detail using an example in Section 4. We do not need to add a dummy round at the beginning of the block cipher to protect against collision fault analysis since an attacker will not be able to find a collision. That is, if an attacker injects a fault in one of the early rounds of a block cipher, the effect of this fault will be propagated by the proposed countermeasure in such a way that an attacker cannot hope to find a collision by making hypotheses on a subset of the secret key bits.

Application to Feistel Ciphers. In Algorithm 4 we define a redundant algorithm for use with Feistel ciphers using the same principles.

Algorithm 4. Redundant Feistel Cipher with Dummy Rounds

Input: P, k_i for $i \in \{1, \ldots, n\}$ (n subkeys from key K), (β, α, k_0).
Output: $C = \texttt{BlockCipher}(P, K)$

State $R_0 \leftarrow P_r$; Redundant State $R_1 \leftarrow P_r$; Dummy state $R_2 \leftarrow \beta$;
State $T_0 \leftarrow P_l$; Redundant State $T_1 \leftarrow P_l$; Dummy state $T_2 \leftarrow \alpha$;
$C_0 \leftarrow 0$; $C_1 \leftarrow 0$; $C_2 \leftarrow \beta$; $i \leftarrow 1$;
while $i \leq 2\,n$ **do**
  $\lambda \leftarrow \texttt{RandomBit}()$; // $\lambda = 0$ implies a dummy round
  $\kappa \leftarrow (i \wedge \lambda) \oplus 2\,(\neg\lambda)$;
  $\zeta \leftarrow \lceil i/2 \rceil\, \lambda$; // ζ is actual round counter, 0 for dummy
  $R_\kappa \leftarrow \texttt{RoundFunction}(R_\kappa, k_\zeta) \oplus T_\kappa$;
  $C_\kappa \leftarrow R_\kappa \oplus C_2$; // infect C_κ to propagate a fault
  $\epsilon \leftarrow \lambda(\neg(i \wedge 1)) \cdot \texttt{SNLF}(C_0 \oplus C_1)$; // check if i is even
  $T_0 \leftarrow T_0 \oplus \epsilon$;
  $R_0 \leftarrow R_0 \oplus \epsilon$;
  $R_2 \leftarrow R_2 \oplus \epsilon$;
  $i \leftarrow i + \lambda$;
end
$C_2 \leftarrow \texttt{RoundFunction}(R_2, k_0) \oplus T_2$;
$R_0 \leftarrow R_0 \oplus C_2$;
return $T_0 \| R_0$

Remark. In our algorithms we assume that the block cipher consists of either an S-P network or a Feistel construction. In both cases it is straightforward to compute a constant pair (β, k_0) resp. triplet (α, β, k_0) by simply inverting the round function and choosing a suitable k_0 for a chosen β (and α).

In defining the countermeasures to protect a secure implementation, one would use a redundant implementation to protect a certain number of rounds at the beginning and the end of the block cipher. That is, one would use a round as defined in Algorithm 1 resp. 2 for rounds that do not require explicit protection and a round as defined in Algorithm 3 resp. 4 for those that do. Further, the dummy rounds should not be distinguishable from every other round by means of side-channel analysis. We detail how this can be achieved in Section 5.

3.3 Performance

We assess the performance overhead of the proposed algorithms, first for S-P networks and then for Feistel ciphers. We begin with determining the cost of making "normal" dummy rounds smart, then we add the cost of making the implementations redundant. We assume that the baseline implementation already

uses dummy rounds, and the analysis is independent of the chosen function `RandomBit`, i.e. of the number of dummy rounds. The number of redundant rounds is also not covered by our analysis.

For S-P networks, the overhead to make dummy rounds smart (Algorithm 1) is: one additional constant (β) the size of the cipher state, and two XOR operations on operands the size of the cipher state in each round. The generation of the random bits, the additional (dummy) rounds, and the second cipher state are already required by using dummy rounds without our countermeasure.

In order to further equip the implementation of the S-P network with smart redundant rounds, as done in Algorithm 3, four additional states, and one round execution followed by an XOR at the end of the implementation are required. In addition, some nonlinear function, four XORs and four logic operations have to be computed for each round.

For Feistel ciphers, the overhead to make dummy rounds smart (Algorithm 2) is: one additional state of half the block size, two constants of half the block size (α and β), and four additional XOR operations on operands half the block size per round.

The fault protection for Feistel ciphers in Algorithm 4 requires five additional states of half the block size, and one round execution followed by two XORs at the end of the implementation. Moreover, each round requires an additional evaluation of some nonlinear function, two XOR operations, and four logic operations. Table 2 shows a summary of the results.

Table 2. Performance overheads: from dummy rounds to smart dummy rounds (left), and from smart dummy rounds to smart dummy rounds with redundancy (right)

	Smart Dummy rounds		+ Redundant Rounds	
	Overall	per round	Overall	per round
S-P Network	+ 1 Const	+ 2 XOR	+ 4 States + 1 XOR + 1 RF	+ 4 XOR + 4 Logic + 1 NLF
Feistel Cipher	+ 1/2 State + 2/2 Const	+ 4 XOR	+ 5/2 States + 2 XOR + 1RF	+ 2 XOR + 4 Logic + 1 NLF

4 Evaluating the Countermeasure

In this section we evaluate the security of the above presented redundant algorithms for S-P networks and Feistel ciphers. We describe how the weakest instance of the presented countermeasure would affect the strongest attacks available. Any other attacks from the literature will typically require a more complex analysis, e.g. a larger fault, more faulty ciphertexts, or a fault further into the algorithm, and the effect of the last dummy round will become more pronounced. Some complex attacks and those requiring particular circumstances, such as collision fault analysis, are not considered since they are completely prevented by the countermeasure.

4.1 S-P Networks

In this section we discuss Algorithm 3 in terms of its resistance to differential fault analysis. To demonstrate this we use the AES as an example, since it will be the most likely instance of our countermeasure in S-P networks.

The Advanced Encryption Standard (AES) [19] was standardized in 2001 from a proposal by Daemen and Rijmen [20]. Note that we restrict ourselves to considering AES-128 and that in discussing the AES we consider that all variables are arranged in a 4×4 array of bytes, known as the state matrix. For example the 128-bit plaintext $P = (p_1, p_2, \ldots, p_{16})_{(256)}$ is arranged as follows:

$$\begin{pmatrix} p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \\ p_4 & p_8 & p_{12} & p_{16} \end{pmatrix}.$$

The encryption itself is conducted by the repeated use of a round function that comprises the following operations executed in sequence:

`SubBytes` — The only nonlinear step of the block cipher, consisting of a substitution table applied to each byte of the state.

`ShiftRows` — A byte-wise permutation of the state that operates on each row.

`MixColumns` — Each column of the state matrix is considered as a vector where each of its four elements belong to $\mathbb{F}(2^8)$. A $4{\times}4$ matrix M whose elements are also in $\mathbb{F}(2^8)$ is used to map this column into a new vector. This operation is applied to the four columns of the state matrix. Here M is defined as

$$M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

where all the elements in M are elements of $\mathbb{F}(2^8)$ expressed in decimal. This function is not included in the last round.

`AddRoundKey` — XORs each byte of the array with a byte from a corresponding subkey. An initial subkey addition precedes the first round.

The simplest, yet strongest, example of differential fault analysis was proposed by Piret and Quisquater [21], where it is assumed that a one byte fault is induced at the beginning of the ninth round. If, for example, the byte at index one is modified by a fault, the difference between the result and what it ought to be at the end of the ninth round becomes:

$$\begin{pmatrix} 2\theta & 0 & 0 & 0 \\ \theta & 0 & 0 & 0 \\ \theta & 0 & 0 & 0 \\ 3\theta & 0 & 0 & 0 \end{pmatrix}$$

An attacker can compare a faulty and the correct ciphertext using the following relationship, using the notation defined above,

$$\begin{aligned} 2\,\theta &= S^{-1}(c_1 \oplus k_1) \oplus S^{-1}(c'_1 \oplus k_1) \\ \theta &= S^{-1}(c_{14} \oplus k_{14}) \oplus S^{-1}(c'_{14} \oplus k_{14}) \\ \theta &= S^{-1}(c_{11} \oplus k_{11}) \oplus S^{-1}(c'_{11} \oplus k_{11}) \\ 3\,\theta &= S^{-1}(c_8 \oplus k_8) \oplus S^{-1}(c'_8 \oplus k_8)\,, \end{aligned}$$

where c'_i and k_i for $i \in \{1, \ldots, 16\}$ represents the bytes from the state matrix of the faulty ciphertext and the last subkey respectively. This will allow the 2^{32} possible key hypotheses for $\{k_1, k_8, k_{11}, k_{14}\}$ to be reduced to 2^8 possibilities. The procedure can then be repeated to derive the entire last subkey.

With our countermeasures in place, the equations would become:

$$\begin{aligned} 2\,\theta &= S^{-1}(c_1 \oplus k_1) \oplus S^{-1}(c'_1 \oplus k_1 \oplus (b_1 \oplus \gamma_1)) \\ \theta &= S^{-1}(c_{14} \oplus k_{14}) \oplus S^{-1}(c'_{14} \oplus k_{14} \oplus (b_{14} \oplus \gamma_{14})) \\ \theta &= S^{-1}(c_{11} \oplus k_{11}) \oplus S^{-1}(c'_{11} \oplus k_{11} \oplus (b_{11} \oplus \gamma_{11})) \\ 3\,\theta &= S^{-1}(c_8 \oplus k_8) \oplus S^{-1}(c'_8 \oplus k_8 \oplus (b_8 \oplus \gamma_8))\,, \end{aligned}$$

where we define b_i for $i \in \{1, \ldots, 16\}$ as the output of the `SNLF`$(C_0 \oplus C_1)$ at the end of the penultimate round, and γ_i for $i \in \{1, \ldots, 16\}$ as the result of the last dummy round. The `SNLF` cannot be implemented as the `SubBytes` operation since it does not map a zero to a zero as required, but one could, for example, use inversion in $\mathbb{F}(2^8)$.

In this example we consider that the `MixColumns` operation is not included to simplify the analysis. However, an equivalent analysis would be straightforward if the dummy round included a `MixColumns` operation. One can simplify the above by rewriting each $b_i \oplus \gamma_i$, for $i \in \{1, 8, 11, 14\}$, as one unknown byte. However, an attacker will not be able to use the equations to reduce the number of possible keys since there will be 2^{48} solutions.

4.2 Feistel Structures

We discuss Algorithm 4 in terms of its resistance to differential fault analysis and use the Data Encryption Standard (DES) as an example. We consider DES relevant since it is still widely used in banking.

DES was introduced by NIST in the mid 1970s [22], and was the first openly available cryptography standard. DES can be considered as a transformation of two 32-bit variables (L_0, R_0), i.e. the message block, through sixteen iterations of the Feistel structure to produce a ciphertext block (L_{16}, R_{16}). The Expansion and P-permutations are bitwise permutations. For clarity of expression, these permutations will not always be considered and the round function for round n will be written as:

$$\begin{aligned} L_n &= R_{n-1} \\ R_n &= S(R_{n-1} \oplus K_n) \oplus L_{n-1} \end{aligned}$$

where S is a nonlinear substitution function.

In this section we describe the strongest attack that can be applied to an implementation of DES. This fault attack on DES involves injecting a fault in the fifteenth round and was described by Biham and Shamir [23]. The last round of DES can be expressed in the following manner:

$$\begin{aligned} R_{16} &= S(R_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(L_{16} \oplus K_{16}) \oplus L_{15}\,. \end{aligned}$$

For ease of expression we ignore the bitwise permutations since they will only impact an implementation of the described attack. If a fault occurs during the execution of the fifteenth round, i.e. R_{15} is randomized by a fault to become R'_{15}, then:

$$\begin{aligned} R'_{16} &= S(R'_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(L'_{16} \oplus K_{16}) \oplus L_{15} \end{aligned}$$

and

$$\begin{aligned} R_{16} \oplus R'_{16} &= S(L_{16} \oplus K_{16}) \oplus L_{15} \oplus S(L'_{16} \oplus K_{16}) \oplus L_{15} \\ &= S(L_{16} \oplus K_{16}) \oplus S(L'_{16} \oplus K_{16})\,. \end{aligned}$$

This provides an equation in which only the last subkey, K_{16}, is unknown. All of the other variables are available from the ciphertext block. This equation holds for each S-box in the last round, which means that it is possible to search for key hypotheses in sets of six bits. This will return an expected 2^{24} key hypotheses for the last round key and, therefore, 2^{32} hypotheses for the block cipher key.

If an attacker attempts to apply this attack to an instance of Algorithm 4, the last dummy round would need to be taken into account. A faulty ciphertext would have the form:

$$\begin{aligned} R'_{16} &= S(R'_{15} \oplus K_{16}) \oplus L_{15} \oplus (b \oplus \gamma) \\ &= S(L'_{16} \oplus K_{16}) \oplus L_{15} \oplus (b \oplus \gamma)\,, \end{aligned}$$

where b denotes the result of the $\texttt{SNLF}(C_0 \oplus C_1)$ function at the end of the last round and γ denotes the result of the last dummy round. We obtain the difference

$$R_{16} \oplus R'_{16} = S(L_{16} \oplus K_{16}) \oplus S(L'_{16} \oplus K_{16}) \oplus (b \oplus \gamma)\,.$$

As in the previous example, this would have too many solutions to provide any information on the last subkey since $b \oplus \gamma$ is unknown.

5 Further Strengthening the Countermeasure

In Algorithm 1 we require that

$$\texttt{RoundFunction}(\beta, k_0) = \beta\,. \tag{1}$$

If the pair (β, k_0) is fixed for a given implementation, the computation during all dummy rounds will be identical. Therefore, the pattern in a given side-channel

generated by a dummy round would have a given form. An attacker could potentially exploit this weakness to identify dummy rounds by cross correlation or template analysis [24].

To solve this problem one can simply refresh the pair (β, k_0) with a frequency determined by how powerful an attacker is assumed to be. More precisely, one would randomly generate β (or k_0) and change k_0 (or β) such that (1) holds. This fully randomizes the computation during a dummy round but it may still be possible to identify a dummy round as a round with equal input and output.

A better, but also more costly, approach would be to generate a triplet of random values (β, k_0, δ) where

$$\texttt{RoundFunction}(\beta, k_0) = \delta\,.$$

The input and output of a dummy round would be random and an attacker would no longer be able to identify a dummy round. However, this approach requires that the effect of the δ be corrected. For example, in Algorithm 1, line 6 would need to be replaced with

$$R_0 \leftarrow R_0 \oplus R_1 \oplus \beta \oplus \delta.$$

This makes the countermeasure more expensive but ensures that an attacker would be unable to identify a dummy round by any means.

The application of this latter idea to Algorithm 2 would require a randomly generated quartet $(\beta, k_0, \alpha, \delta)$, since we would require that

$$\texttt{RoundFunction}(\beta, k_0) \oplus \alpha = \delta\,.$$

Otherwise, strengthening the countermeasure for Feistel ciphers would be similar to strengthening that for S-P networks in Algorithm 1.

6 Conclusion

In this paper we describe algorithms where dummy and redundant rounds can be used to implement infective computation in block ciphers. This would allow a block cipher to be implemented where no check is required to detect whether a fault has occurred since no information would be available to an attacker. This would prevent an attacker from implementing a multiple-fault attack that affects tests at the end of a block cipher [25,18].

We have demonstrated that the algorithms will be secure against the strongest available differential fault analysis that can be applied to AES and DES. We have not demonstrated this for all fault attacks that have been described in the literature since many attacks will not be possible. For example, collision fault analysis requires an attacker to inject a fault in one of the first rounds of a block cipher to produce a faulty ciphertext. The attacker will then try to find a plaintext that will produce the same ciphertext without a fault. Given that such a fault will re-infect the implementation numerous times, such an attack becomes impossible.

The current state-of-the-art in implementing block ciphers is to implement some form of consistency check in order to detect faults, so that the output can be withheld if a fault has occurred. This is potentially vulnerable to an attacker who can inject the same fault into both redundant paths of the algorithm, thus by-passing the check at the end of the algorithm. This would be a complex attack, but the verification would aid an attacker since a faulty result would only appear once the fault injection is successful. With our countermeasures an attacker will always receive a ciphertext. Thus, an adversary has to analyze the output without knowing whether the fault injection was successful and the output contains exploitable information, or not. Given that injecting two identical faults into a single run of an unknown implementation with random dummy rounds is very hard and will only succeed with a very low probability, it would be very difficult to determine at what point a sufficient number of exploitable faulty ciphertexts have been collected, and to distinguish them from the ones that do not provide any information on the secret key.

The use of the proposed countermeasures will also help to avoid a situation where a new fault attack is published that allows an attacker to exploit faults over more rounds than that considered necessary when a given algorithm is implemented. This is because such attacks will typically require a relatively large number of faulty ciphertexts [26,27], and our countermeasure will insert faulty ciphertexts containing no information into the analysis. This will hinder or prevent new attacks.

Acknowledgements. The work described in this paper has been supported in part by the European Commission through the ICT Programme under contract ICT-2007-216676 ECRYPT II and under contract ICT-SEC-2009-5-258754 TAMPRES, by the EPSRC via grant EP/I005226/1, by the Research Council of KU Leuven: GOA TENSE (GOA/11/007), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the Flemish Government FWO G.0550.12N and by the Hercules Foundation AKUL/11/19. Benedikt Gierlichs is Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

References

1. 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)
2. 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)
3. 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)
4. 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)

5. 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)
6. 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)
7. Goubin, L., Patarin, J.: DES and Differential Power Analysis: the "Duplication" Method. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999)
8. Blömer, J., Seifert, J.-P.: Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, pp. 162–181. Springer, Heidelberg (2003)
9. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
10. Clavier, C.: Secret External Encodings Do Not Prevent Transient Fault Analysis. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 181–194. Springer, Heidelberg (2007)
11. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 224–233. Springer, Heidelberg (2011)
12. Yen, S.-M., Joye, M.: Checking Before Output Not be Enough Against Fault Based Cryptanalysis. IEEE Transactions on Computers 49(9), 967–970 (2000)
13. Karpovsky, M.G., Kulikowski, K.J., Taubin, A.: Robust Protection Against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard. In: International Conference on Dependable Systems and Networks (DSN 2004), pp. 93–101. IEEE (2004)
14. Genelle, L., Giraud, C., Prouff, E.: Securing AES Implementation Against Fault Attacks. In: Naccache, D., Oswald, E. (eds.) Fault Diagnosis and Tolerance in Cryptography, pp. 51–62. IEEE (2009)
15. Malkin, T., Standaert, F.-X., Yung, M.: A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 159–172. Springer, Heidelberg (2006)
16. Medwed, M., Schmidt, J.-M.: A Continuous Fault Countermeasure for AES Providing a Constant Error Detection Rate. In: Breveglieri, L., Joye, M., Koren, I., Naccache, D., Verbauwhede, I. (eds.) Fault Diagnosis and Tolerance in Cryptography, pp. 66–71. IEEE (2010)
17. Kim, C.H., Quisquater, J.-J.: Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) WISTP 2007. LNCS, vol. 4462, pp. 215–228. Springer, Heidelberg (2007)
18. van Woudenberg, J.G.J., Witteman, M.F., Menarini, F.: Practical Optical Fault Injection on Secure Microcontrollers. In: Breveglieri, L., Guilley, S., Koren, I., Naccache, D., Takahashi, J. (eds.) Fault Diagnosis and Tolerance in Cryptography, pp. 91–99. IEEE (2011)
19. FIPS PUB 197: Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA (2001)
20. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. In: AES Round 1 Technical Evaluation CD-1: Documentation, NIST (August 1998), http://www.nist.gov/aes

21. Piret, G., Quisquater, J.-J.: A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Heidelberg (2003)
22. NIST: Data Encryption Standard (DES) (FIPS–46-3). National Institute of Standards and Technology (1999)
23. 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)
24. 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. 172–186. Springer, Heidelberg (2003)
25. Kim, C.H., Quisquater, J.-J.: New Differential Fault Analysis on AES Key Schedule: Two Faults Are Enough. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 48–60. Springer, Heidelberg (2008)
26. Derbez, P., Fouque, P.-A., Leresteux, D.: Meet-in-the-Middle and Impossible Differential Fault Analysis on AES. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 274–291. Springer, Heidelberg (2011)
27. Rivain, M.: Differential Fault Analysis on DES Middle Rounds. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 457–469. Springer, Heidelberg (2009)

Author Index