Frederik Armknecht
Stefan Lucks (Eds.)

LNCS 7242

Research in Cryptology

4th Western European Workshop, WEWoRC 2011
Weimar, Germany, July 2011
Revised Selected Papers

Springer

Lecture Notes in Computer Science 7242

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

Frederik Armknecht Stefan Lucks (Eds.)

Research in Cryptology

4th Western European Workshop, WEWoRC 2011
Weimar, Germany, July 20-22, 2011
Revised Selected Papers

Springer

Volume Editors

Frederik Armknecht
Universität Mannheim
Arbeitsgruppe Theoretische Informatik und IT-Sicherheit
A5,6, 68131 Mannheim, Germany
E-mail: armknecht@uni-mannheim.de

Stefan Lucks
Bauhaus-Universität Weimar
Fakultät Medien
Bauhausstraße 11, 99423 Weimar, Germany
E-mail: stefan.lucks@uni-weimar.de

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-34158-8 e-ISBN 978-3-642-34159-5
DOI 10.1007/978-3-642-34159-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012948763

CR Subject Classification (1998): E.3, D.4.6, I.1, K.6.5, K.4.4, G.2.1

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

The Western European Workshop on Research in Cryptology (WEWoRC) is a bi-annual workshop with a specific focus on research performed by junior scientists. The specific focus is exhibited by the rule that at least one of the authors of a paper to be presented and, especially, included in the proceedings must be a junior researcher.

WEWoRC 2011 was held at the Bauhaus-Universität Weimar in Germany, organized by the Department of Media, Bauhaus-Universität Weimar, in cooperation with the Gesellschaft für Informatik e.V. (GI). It was the fourth of its kind, after WEWoRC 2005 in Leuven, Belgium, WEWoRC 2007 in Bochum, Germany, and WEWoRC 2009 in Graz, Austria.

Beyond the rule of requiring at least one junior researcher to (co-) author a paper, the WEWoRC is special because:

1. The authors submit a short abstract for WEWoRC.
2. If the Program Chairs consider the paper on topic for WEWoRC, the authors are asked to present their work at the workshop.
3. At the workshop, the Program Chairs invite the best presenters and the authors of the best abstracts to submit a full version (or an extended abstract) for the final proceedings.
4. These submissions are reviewed by the Program Committee(PC), which eventually decides which will be included in the proceedings.

That means unfinished ideas and, sometimes, unpublishable work can still be presented at the workshop, while the proceedings will only include mature work and good results. We believe that this process addresses the specific needs of junior researchers better than the traditional all-or-nothing approach at cryptographic research meetings, where the review comes first and papers are either presented and published in the proceedings, or neither presented nor published.

The technical program of WEWoRC 2011 consisted of 25 submitted and two invited talks. The invited talks where given by two senior researchers, Heike Neumann from NXP Semiconductors on "The Practice of Cryptography" and Marc Fischlin from TU Darmstad on "Key Exchange – 35 Years and Still Rolling." The technical program was amended by a social program: A welcome reception on the evening before the conference, the "Bauhaus Walk" at the places where the famous Bauhaus school for crafts and fine arts was founded, and a conference dinner.

After being invited for the final proceedings and reviewed by at least four members of the PC (submissions with a member from the PC as one of the authors where reviewed by six independent PC members), ten papers where finally chosen for these proceedings.

Thank you! We thank all the authors and co-authors of abstracts submitted to WEWoRC for presentation, and, especially all the junior researchers and the invited speakers, who came to Weimar to present their work and participate in the discussions. We are grateful to the the local staff from Bauhaus-Universität Weimar, who worked hard to make WEWoRC 2011 possible and successful (in alphabetical order: Christian, Ewan, Jakob, Nadin, Theresa). We thank the PC members for their reviews and lively discussions. Last, but not least, we thank the sponsors, NXP Semiconductors and NEC Europe, whose contribution allowed us to support some ill-funded presenters and to invite the speakers for the invited talks.

July 2012

Frederik Armknecht
Stefan Lucks

Organization

Program Committee

F. Armknecht
J.-M. Bohli
C. Boyd
C. Cid
O. Dunkelman
J. von zur Gathen
W. Geiselmann
H. Handschuh
F. Hess
S. Katzenbeisser
G. Leander
S. Lucks
R. Maes
A. May
W. Meier
H. Neumann
T. Peyrin
B. Preneel
C. Rechberger
V. Rijmen
A.-R. Sadeghi
J.-P. Seifert
F.-X. Standaert
M. Yung
C. Wolf
E. Zenner

Sponsoring Institutions

NXP Semiconductors N.V., Eindhoven, The Netherlands
NEC Europe Ltd., Heidelberg, Germany

Table of Contents

Broadcast Attacks against Code-Based Schemes

Robert Niebuhr[1] and Pierre-Louis Cayrel[2]

[1] Technische Universität Darmstadt
Fachbereich Informatik
Kryptographie und Computeralgebra,
Hochschulstraße 10
64289 Darmstadt
Germany
rniebuhr@cdc.informatik.tu-darmstadt.de
[2] Laboratoire Hubert Curien, UMR CNRS 5516,
Bâtiment F 18 rue du Professeur Benoît Lauras
42000 Saint-Etienne
France
pierre.louis.cayrel@univ-st-etienne.fr

Abstract. Code-based cryptographic schemes are promising candidates for post-quantum cryptography since they are fast, require only basic arithmetic, and because their security is well understood. While there is strong evidence that cryptosystems like McEliece and Niederreiter are secure, they have certain weaknesses when used without semantic conversions. An example is a broadcast scenario where the same message is send to different users, encrypted with the respective keys.

In this paper, we show how an attacker can use these messages to mount a broadcast attack, which allows to break the Niederreiter and the HyMES cryptosystem using only a small number of messages. While many code-based cryptosystems use certain classes of codes, e.g. binary Goppa codes, our attack is completely independent from this choice and solves the underlying problem directly. Since the number of required messages is very small and since the attack is also possible if related, not identical messages are sent, this has many implications on practical cryptosystem implementations. We discuss possible countermeasures, and provide a CCA2-secure version of the Niederreiter cryptosystem using the Kobara-Imai conversion.

Keywords: Broadcast Attack, Niederreiter, McEliece, codes, post quantum, cryptography.

Introduction

In 1988, J. Håstad [9] presented an attack against public key cryptosystems. This attack was originally aimed at the RSA cryptosystem, when a single message is sent to different recipients using their respective public keys. Håstad showed how to recover the message in this broadcast scenario. While this result is known for

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 1–17, 2012.

a long time, this type of attack has not been considered for cryptosystems based on error-correcting codes.

Our Contributions

In the following, we show how and under what conditions an attacker can mount a broadcast attack against two code-based encryption schemes: the Niederreiter and the HyMES cryptosystem. Our attack allows to recover the secret message. We show that if the public keys corresponding to the intercepted messages are independent from each other, we expect to require no more than N_r recipients to run the attack:

$$N_r := \left\lceil \frac{n+2}{r} \right\rceil,$$

where n and $r = n-k$ denote the code parameters of the users' secret keys. That means that with near certainty, $N \geq N_r$ recipients suffice to run the attack. Table 1 shows the number of required recipients for selected parameter sets taken from Bernstein et al. [2] and Biswas [5]. In most cases, a very small number of identical (or similar) message is sufficient to completely break the schemes. We achieve this by combining the intercepted information into a large set of linear equations until the system has full rank and can be solved.

In addition to that, we treat the cases when the attacker receives less messages than required for this attack, and when the cleartexts are related, instead of identical. In the former case, the attack complexity is higher compared with the broadcast attack before, but lower than a generic attack: after setting up the linear equation system, the attacker runs an ISD attack, the complexity of which is smaller the more messages are intercepted. In the latter case, more messages are required to perform the broadcast attack: $N'_r := \lceil \frac{n+2}{r-(u+2)} \rceil$, where u denotes the number of bits where not all messages are identical to each other.

While many code-based cryptosystems use certain classes of codes, e.g. binary Goppa codes, our attack is completely independent of this choice and solves the underlying problem directly. That means, no matter what class of codes is used, the attack complexity cannot be greater than our results; it might be possible, though, to achieve even better results against certain classes by exploiting the structure of the code. To illustrate our results, we apply our broadcast attack implementation against the Niederreiter cryptosystem in the FlexiProvider package [6], recovering the message in only a few seconds to minutes (see Table 2 on page 15). We conclude this section with a discussion on possible countermeasures and provide a CCA2-secure version of the Niederreiter cryptosystem using the Kobara-Imai conversion.

Related Work

Our attack is related to the one by Plantard and Susilo [15] who studied broadcast attacks against lattice-based schemes. Our analysis, however, is more

thorough since we prove an explicit bound for the expected number of recipients that is required to run our attack. We also cover the case where too few messages are intercepted by an attacker and analyze the situation where the broadcasted messages are not identical but similar to each other. Finally, we discuss in detail the use of semantic conversions to protect against broadcast attacks. An implementation of our attack successfully demonstrated its efficiency, even though it is in Java and not at all optimized for speed.

A recent paper [18] by Pan and Deng presents a broadcast attack against the NTRU cryptosystem. The authors use a Learning with Errors (LWE) algorithm by Ding for their attack.

Organization of the Paper

In Section 1 we begin with a review on code-based encryption schemes. The next section covers our broadcast attack, including the two variants of insufficient number of messages and related-messages. We discuss possible countermeasures in Section 3. In the subsequent Section 4 we present our implementation and provide numerical results. We conclude in Section 5.

1 Code-Based Encryption Schemes

Code-based cryptographic schemes are based on the difficulty of two hard problems: Code distinguishing and syndrome decoding. In this paper, we will focus on the latter, which is defined as follows:

Problem 1 (Syndrome-decoding problem). Given a matrix H and a vector c, both over $\mathbb{F}_q$, and a non-negative integer t; find a vector $x \in \mathbb{F}_q^n$ of (Hamming) weight t such that $Hx^T = c^T$.

Among these are the McEliece and Niederreiter cryptosystems. They are both encryption schemes, and the latter is the basis for the CFS signature scheme. Another cryptosystem we analyze in this paper is HyMES (Hybrid McEliece Encryption Scheme)[1], which uses techniques from both schemes above: It encrypts similar to the McEliece scheme, but encodes part of the messages in the error vector, similar to the Niederreiter scheme.

In this section, we will briefly describe these three cryptosystems. In the following, let G be a $k \times n$ generator matrix for an $(n, k = n - r, t)$ Goppa code $\mathcal{C}$, H be a corresponding $r \times n$ parity check matrix, and c a vector of length r. Let the message m be a vector of length k, and φ a bijective function mapping an integer to a word of length n and weight t. All matrices and vectors are defined over a finite field $\mathbb{F}_q$, where q is a prime power.

[1] http://www-rocq.inria.fr/secret/CBCrypto/index.php?pg=hymes

Notation. In our algorithms, we use the following notation:

$\mathrm{wt}(v)$	: The (Hamming) weight of vector v
$\mathrm{PRG}(x)$	: Cryptographically secure pseudo random number generator
l	: $\lfloor \log_q \binom{n}{t} \rfloor$
$\varphi()$	: Bijective function mapping an integer in $\mathbb{Z}_{q^l} = \mathbb{Z}/q^l\mathbb{Z}$ to a word of length n and weight t. We apply φ to vectors in $\mathbb{F}_q^k$ by enumerating these vectors first, and then apply φ
$\mathrm{MSB}_x(v)$	: The left x bits of v
$\mathrm{LSB}_x(v)$	: The right x bits of v
$\mathrm{len}(v)$	: Length of vector v
$h()$	: Cryptographic secure hash function to a word of length l

Additionally, we write $A = (B|C)$ and $A' = \langle B, C \rangle$ to denote the horizontal and vertical concatenation of B and C, respectively, where these can be vectors or matrices. For a matrix A and $J \subseteq \{1, 2, \ldots n\}$, $A_{.J}$ denotes the submatrix of A consisting of those columns indexed by J.

1.1 McEliece

The McEliece public-key encryption scheme was presented by R. McEliece in 1978 [12]. The original scheme uses binary Goppa codes, for which it remains unbroken (with suitable parameters), but the scheme can be used with any class of codes for which an efficient decoding algorithm is known.

1.2 Niederreiter

In 1986, H. Niederreiter proposed a cryptosystem [14] which can be seen as dual to the McEliece scheme. It uses the parity check matrix of a (usually binary Goppa) code to compute the syndrome of the message, which serves as the ciphertext. Even though the Niederreiter cryptosystem has been proven equally secure as the McEliece system [11], it is threatened by broadcast attacks.
Since the underlying Goppa code can only correct a certain number $t < n$ of errors, the Niederreiter scheme uses a function φ which maps the message to a word of weight t, which is then encrypted.

1.3 HyMES

The HyMES hybrid McEliece cryptosystem developed by N. Sendrier and B. Biswas increases the efficiency of the McEliece scheme by encoding part of the message into the error vector. While in the usual scenario this scheme is as secure as the original McEliece scheme, we will show that it is vulnerable to a broadcast attack.

The HyMES scheme works as follows: The message m is split into two parts $m = (m_1|m_2)$. The first part m_1 corresponds to the message in the original

Algorithm 1. The McEliece cryptosystem

Notation for Algorithm 1:

G : A $k \times n$ generator matrix
P : An $n \times n$ random permutation matrix
S : A $k \times k$ invertible matrix
$\mathcal{D}_G$: A decoding algorithm for the underlying (n, k, t) code $\mathcal{C}$

Encryption $\mathbf{Enc}^{\mathbf{McEliece}}$
INPUT: Message $m \in \mathbb{F}_q^k$ and random seed $r \in \{0,1\}^*$
OUTPUT: Ciphertext $c \in \mathbb{F}_q^n$

$\widehat{G} \leftarrow SGP$
$e \leftarrow \mathrm{PRG}(r)$, such that $\mathrm{wt}(e) = t$
$c \leftarrow m\widehat{G} + e$

Return c

Decryption $\mathbf{Dec}^{\mathbf{McEliece}}$
INPUT: Ciphertext c
OUTPUT: Message m

$\widehat{c} \leftarrow cP^{-1} = mSG + eP^{-1}$
$mSG \leftarrow \mathcal{D}_G(\widehat{c})$
▷ Let $J \subseteq \{1, \ldots, n\}$ be a set such that $G_{\cdot J}$ is invertible
$m \leftarrow mSG \cdot G_{\cdot J}^{-1} \cdot S^{-1}$

Return m

Algorithm 2. The Niederreiter cryptosystem

Notation for Algorithm 2:

H : A $r \times n$ parity check matrix
$\mathcal{D}_H$: A decoding algorithm for the underlying $(n, k = n - r, t)$ code $\mathcal{C}$

Encryption $\mathbf{Enc}^{\mathbf{N}}$
INPUT: Message $m \in \mathbb{F}_q^l$
OUTPUT: Ciphertext $c \in \mathbb{F}_q^r$

$c \leftarrow H \cdot \varphi(m)^T$

Return c

Decryption $\mathbf{Dec}^{\mathbf{N}}$
INPUT: Ciphertext c
OUTPUT: Message m

$m \leftarrow \varphi^{-1}(\mathcal{D}_H(c))$

Return m

McEliece scheme, while the second part is encoded into a word of weight t and serves as the error vector $e = \varphi(m_2)$. There are many possible encoding functions φ, e.g. enumerative encoding or encoding into regular words, but the choice of φ is not relevant in our context.

Algorithm 3. The HyMES cryptosystem

Notation for Algorithm 3:
- G : A $k \times n$ generator matrix
- P : An $n \times n$ random permutation matrix
- S : A $k \times k$ invertible matrix
- $\mathcal{D}_G$: A decoding algorithm for the underlying (n, k, t) code $\mathcal{C}$

Encryption $\mathbf{Enc^{HyMES}}$
INPUT: Message $m \in \mathbb{F}_q^{k+l}$
OUTPUT: Ciphertext $c \in \mathbb{F}_q^n$

$\widehat{G} \leftarrow SGP$
$m_1 \leftarrow \mathrm{MSB}_k(m)$
$m_2 \leftarrow \mathrm{LSB}_l(m)$
$c \leftarrow m_1\widehat{G} + \varphi(m_2)$

Return c

Decryption $\mathbf{Dec^{HyMES}}$
INPUT: Ciphertext c
OUTPUT: Message m

$\widehat{c} \leftarrow cP^{-1} = m_1SG + \varphi(m_2)P^{-1}$
$mSG \leftarrow \mathcal{D}_G(\widehat{c})$
▷ Let $J \subseteq \{1, \ldots, n\}$ be a set such that $G_{.J}$ is invertible
$m_1 \leftarrow mSG \cdot G_{.J}^{-1} \cdot S^{-1}$
$m_2 \leftarrow \varphi^{-1}(c - m_1\widehat{G})$

Return $(m_1|m_2)$

2 Broadcast Attacks against Niederreiter/HyMES

In this section, we will show how to mount a broadcast attack against the Niederreiter and HyMES schemes. The problem solved by a broadcast attack is the following:

Problem 2 (Broadcast attack). Given N ciphertexts c_i of the same message m, encrypted using N corresponding public keys G_i (HyMES) or H_i (Niederreiter), find m.

Both the McEliece and the Niederreiter cryptosystem rely on the hardness of the decoding prob- lem, i.e. finding a codeword in a certain distance to a given

word. The main difference is that in McEliece the cleartext determines the codeword, and the error vector is random, while Niederreiter works essentially vice versa. This difference results in a weakness of the McEliece cryptosystem, the malleability of its ciphertexts: Adding rows of the public key to a ciphertext results in a new valid ciphertext.

Another implication is McEliece's weakness to message-resend and related-message attacks. Two ciphertexts of messages m_1 and m_2, where the messages have a known relation (or are identical), allow an attacker to easily find at least k error-free positions, which allows efficient guessing of error bits. See [2] for more details. The Niederreiter cryptosystem, however, is not vulnerable to these attacks. It is interesting to note that, facing a broadcast attack, the situation is reversed.

2.1 Attacking Niederreiter

Niederreiter ciphertexts are generated by computing $c_i^T = H_i\varphi(m)^T$. Attempting to solve this equation for m can be done by solving the corresponding linear equation system with n variables and r equations. Since $\varphi(m)$ is identical in all computations, we can (vertically) append the matrices $H = \langle H_1, \ldots, H_N\rangle$ and the syndromes $c = (c_1|\ldots|c_N)$. The number of variables stays constant, whereas the number of equations increases. Some of the new equations might be combinations of old ones, so the new number of equations can be smaller than Nr. In Section 2.3, we will show that the number of redundant equations is very small. Since usually $n \approx 2r$, we need very few c_i to increase the rank of H to n, at which point we can solve the system. We will compute the expected number of messages required to break the system in Section 2.3.

Algorithm 4. Broadcast attack against the Niederreiter cryptosystem

INPUT: Parity check matrices $H_i \in \mathbb{F}_q^{r\times n}$ and corresponding ciphertexts $c_i \in \mathbb{F}_q^r$ for $i \in \mathbb{Z}_N$, and finite field $\mathbb{F}_q$
OUTPUT: Message $m \in \mathbb{F}_q^n$

$H \leftarrow \langle H_1, \ldots, H_N\rangle$
$c \leftarrow (c_1|\ldots|c_N)$

Solve the linear equation system $Hm^T = c^T$ over $\mathbb{F}_q$

Return m

Remark 1. There is a different way to describe our attack, seen from another perspective: By adding c_i as the $(n+1)$-th column of H_i (for all i), we add $(\varphi(m)|(q-1))$ as a codeword to every code, since

$$(H_i|c_i)\cdot(\varphi(m)|(q-1))^T = H_i\cdot\varphi(m)^T + (q-1)c_i = qc_i = 0.$$

The message can be found by intersecting these new codes. There are several implementations to compute the intersection of codes, e.g. in Magma. However, we have chosen the approach above since it allows easier understanding of the required number of recipients.

2.2 Attacking HyMES

While the HyMES implementation does provide methods for padding, a technique used to prevent attacks that exploit relations between cleartexts and/or ciphertexts, this method seems to be a placeholder, since it does not add any security.

Similar to our description of the scheme above, in the broadcast scenario we have

$$c_i = m_1 G_i + \varphi(m_2)$$

as the ciphertexts, where $m = (m_1|m_2)$ and $i = 1 \dots N$. Attacking this scheme can be done by finding $\varphi(m_2)$, since this allows to find m_1 by simple linear algebra. We can reduce this problem to the Niederreiter case:

First, the attacker uses the matrices G_i to compute the respective parity check matrices H_i. One way to do this is to compute the *standard form* of G_i, $G_i' = UG_iQ = (I_k|R)$, where I_k is the identity matrix of size k. Then $H_i' = (-R^T|I_{n-k})$ and $H_i = H_i'Q^{-1}$.

Then the attacker computes the syndromes $s_i = H_i \cdot c_i^T$. Since

$$s_i = H_i(m_1 G_i + \varphi(m_2))^T = H_i \cdot \varphi(m_2)^T,$$

we have reduced the problem to the Niederreiter case above.

Algorithm 5. Broadcast attack against the HyMES cryptosystem

INPUT: Generator matrices G_i and corresponding ciphertexts c_i, for $i \in \mathbb{Z}_N$, and finite field $\mathbb{F}_q$
OUTPUT: Message $m \in \mathbb{F}_q^n$

$\forall i \in \mathbb{Z}_n$ perform the following computations
    Find U_i and Q_i such that $G_i' = U_i G_i Q_i = (I_k|R_i)$
    $H_i' \leftarrow (-R^T|I_{n-k})$
    $H_i \leftarrow H_i' Q_i^{-1}$
    $s_i \leftarrow H_i \cdot c_i^T$

$H \leftarrow \langle H_1, \dots, H_N \rangle$
$c \leftarrow (c_1| \dots |c_N)$

Solve the linear equation system $Hm^T = c^T$ over $\mathbb{F}_q$

Return m

Remark 2. As we noted above, the McEliece cryptosystem does not show a vulnerability to broadcast attacks as Niederreiter/HyMES do. We can sum the information contained in the ciphertexts and public keys into a single equation by (horizontal) concatenation: $G = (G_1|\dots|G_N)$, $c = (c_1|\dots|c_N)$, $e = (e_1|\dots|e_N)$, and writing $c = mG + e$.

This code has length nN, dimension k and minimum distance dN, the weight of e is tN. While this concatenated code is somewhat weaker than the original codes, this can be compensated by larger parameters, and it shows no structural weakness like in the Niederreiter case. Figure 1 shows the work factor to break the concatenated code using an Information Set Decoding (ISD) based attack.

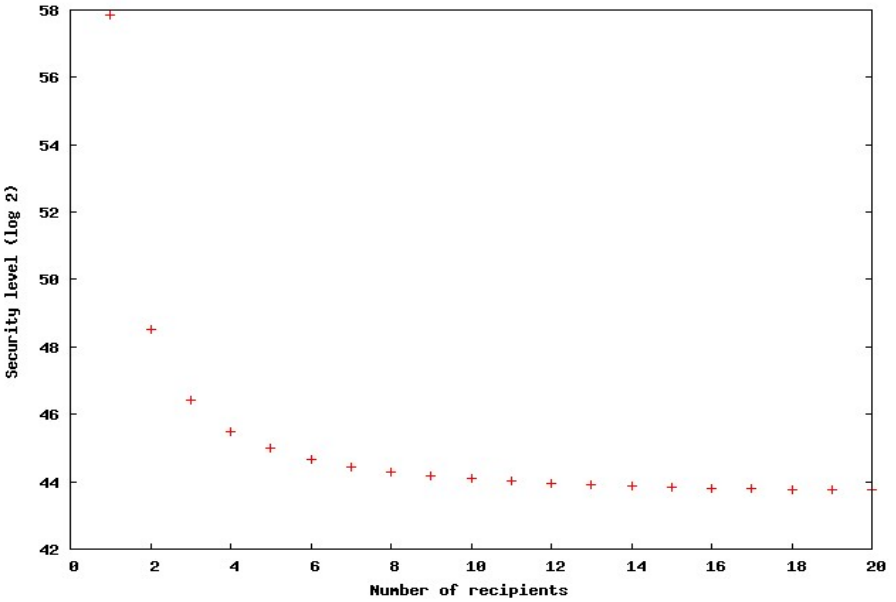


Fig. 1. Work factor to perform an ISD attack against the McEliece cryptosystem with parameters $(n, k, t) = (1024, 524, 50)$ using an increasing number of broadcasted messages

2.3 Expected Number of Recipients Required to Break the Niederreiter/HyMES Scheme

In order to estimate the number of recipients N_r and thus encrypted messages we need to recover the message encrypted by the above Niederreiter/HyMES

schemes, we first assume that the parity check matrices H_i are actually random matrices of full rank (even though it has been shown recently in [7] that we can distinguish random codes from binary Goppa codes for certain parameters, e.g. CFS parameters). We will do the analysis using the Niederreiter scheme, since we reduced the HyMES problem to this one.

We start by estimating the probability that a random vector x is linearly independent from all vectors in A, where A is a given set of r linearly independent vectors over $\mathbb{F}_q$.

A vector is linearly independent from a set of vectors if it cannot be expressed by a linear combination of these vectors. There are q^r (different) linear combinations of vectors in A, and the whole space has dimension q^n. Thus, the probability that x is linearly independent from A is

$$\mathfrak{P} = 1 - q^{r-n} = \frac{q^n - q^r}{q^n}. \tag{1}$$

Therefore, when we start from the system of linear equations

$$H_1 \cdot \varphi(m)^T = c_1^T$$

and add the first row of H_2 to H_1, with probability $\mathfrak{P} = 1 - q^{r-n}$ we add a new equation to the system. Hence, if we assume the vectors in H_2 to be independent from each other, we expect to add $\mathfrak{P}^{-1}$ rows to get one new equation. The fact that the vectors in H_2 are not independent from each other does in fact slightly increases the chance to find a new equation: subsequent vectors in H_2 are guaranteed to be linearly independent to the previous ones already considered, so a small subset of undesired vectors is excluded.

Thus, in order to increase the number of linearly independent equations to n, which allows us to solve the system, we need to add

$$T = \sum_{i=r}^{n-1} \frac{q^n}{q^n - q^i}$$

rows on average.

The codes used in the two cryptosystems are not random, but Goppa codes. Since every non-zero vector in $\mathbb{F}_q^n$ has the same probability to be contained in a Goppa code chosen uniformly at random, the probability $\mathfrak{P}$ and thus the subsequent arguments above remain valid. For example, for the Niederreiter parameters $[n, k] = [1024, 644]$, we need to add 646 rows, which corresponds to 3 recipients.

The expected number D of linearly dependent rows encountered when setting up the system is nearly constant: We add T rows in order to be able to solve the system, out of which $(n - r)$ are not redundant (they complement the initial r rows to a linearly independent set of n rows), and hence

$$\begin{aligned} D &= \sum_{i=r}^{n-1} \frac{q^n}{q^n - q^i} - (n-r) \\ &= \sum_{i=r}^{n-1} \left(\frac{q^n}{q^n - q^i} - 1 \right) \\ &= \sum_{i=r}^{n-1} \frac{q^i}{q^n - q^i} \\ &= \sum_{i=1}^{n-r} \frac{1}{q^i - 1} < 1.7. \end{aligned}$$

The last sum converges quickly, and it decreases with increasing q:

$$\sum_{i=1}^{n-r} \frac{1}{q^i - 1} \leq \sum_{i=1}^{\infty} \frac{1}{2^i - 1} = \sum_{i=1}^{4} \frac{1}{2^i - 1} + \sum_{i=5}^{\infty} \frac{1}{2^i - 1} \leq \sum_{i=1}^{4} \frac{1}{2^i - 1} + \sum_{i=4}^{\infty} \frac{1}{2^i} < 1.7$$

Therefore, we have

$$N_r = \left\lceil \frac{n+2}{r} \right\rceil.$$

Table 1 shows the number of required recipients for selected parameter sets taken from Bernstein et al. [2,3,4] and Biswas [5]. The first column shows the cryptosystem for which the parameters were developed. The number of required recipients, however, does not depend on whether the parameters are used with the Niederreiter or the HyMES cryptosystem.

Table 1. Number of required recipients for Niederreiter and HyMES parameter sets

Cryptosystem	n	k	q	Number of recipients
Niederreiter	2048	1696	2	6
	2048	1608	2	5
	4096	3832	2	16
	4096	3556	2	8
HyMES	1024	524	2	3
	2048	1608	2	5
	2048	1696	2	6
	4096	3604	2	9
	8192	7815	2	22
Niederreiter	3009	2325	2	5
	1931	1491	5	5
	1608	1224	7	5
	1696	1312	9	5
McEliece	1616	1253	2	5
	2928	2244	2	5
	6544	5010	2	5

2.4 Performing a Broadcast Attack When $N < N_r$

When an attacker receives less than N_r broadcast messages, the resulting system $H \cdot x^T = c^T$ is under-determined. It can be used nonetheless to mount an ISD attack. There are different ISD-like attacks, but the basic steps are as follows:

1. Choose a random $n \times n$ permutation matrix Q and compute $H' = QH$.
2. Perform Gaussian elimination on H' and s to get $(I_K|R) = s'$, where I_K is the identity matrix of size K, and $n - K$ is the number of rows of H.
3. Search for $p \leq t$ columns of R such that their sum S has Hamming distance $t - p$ to the syndrome s.
4. The non-zero entries of $S - s$ locate the remaining $t - p$ entries of $\varphi(m)$.

The work factor of this attack can be computed using the formulae in [13]. Figure 2 shows the corresponding attack complexity.

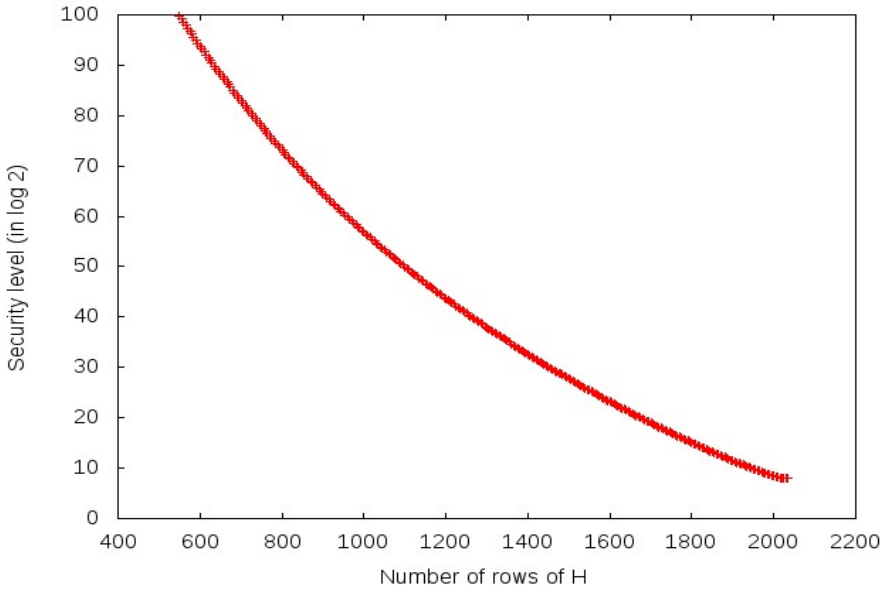


Fig. 2. Work factor for a broadcast attack against McEliece with parameters $(n, k, t) = (1024, 524, 50)$ using ISD when $N < N_r$

This result is supported by [17], where N. Sendrier points out that ISD-based algorithms have an approximate complexity of

$$C(n, R) = a(n) \cdot 2^{-t \cdot \log_2(1-R)},$$

where $R = k/n$ is the information rate and $a(n)$ a polynomial in n. Increasing the number of rows in the parity check matrix decreases R, so $C(n, R)$ decreases exponentially.

2.5 Related-Message Broadcast Attack

In the previous sections, the message m sent to all recipients has been identical. In the following, we show how a broadcast attack can be performed if the messages are *not* identical, but *related*. More concretely, let $M = \{m_i : i = 1..N\}$ and we define the following property:

Definition 1. *A set M of messages $m_i \in \mathbb{F}_q^n$ is called b-related if there are exactly $b \leq n$ coordinates such that all messages are identical on these coordinates. We denote this property by*

$$\rho(M) = b.$$

Since the Niederreiter and HyMES cryptosystems do not encrypt $m_i \in M$ directly, but $\varphi(m_i)$ instead, the choice of the encoding function φ will influence $\rho(\varphi(M))$. To keep our analysis independent of the choice of φ, we will assume that

$$\rho(\varphi(M)) = b,$$

where $n-b$ is small (this is the case, for instance, if φ is a regular words encoding).

Solving the Linear Equation System. For simpler notation, let the messages in M be identical on the left b bits, and (potentially) different on the rightmost $u := n - b$ bits. Since the messages are not identical, the parity check matrices cannot be used directly to form the final system of equations.

Let

$$H_i = (H_i^1 | H_i^2), \quad i = 1..N,$$

where H_i^1 contains the leftmost b columns of H_i.

For 2 recipients, we have the following system of equations

$$\begin{pmatrix} H_1^1 & H_1^2 & 0 \\ H_2^1 & 0 & H_2^2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix},$$

and similarly for N recipients.

A solution $(e_0|e_1|\ldots|e_N)$ (where e_0 has length b, and the other blocks have size $n - b$) yields solutions to the original problem with $m_i = (e_0|e_i)$.

In contrast to the identical-message broadcast attack above, every additional recipient adds equations as well as unknowns to the system. The system will eventually be solvable if (and only if) the number of new equations is greater than the number of new variables. Since we expect a total of at most 2 linearly dependent rows for the system, it is sufficient if $r > n - b + 2 = u + 2$.

The number of recipients N_r' required to solve the system of a related-message broadcast attack is

$$N_r' = \left\lceil \frac{n+2}{r-(u+2)} \right\rceil.$$

3 Countermeasures

Our broadcast attack exploits the fact that the received ciphertexts are related since they correspond to the same message m. A similar fact is used in other types of attack like message-resend, related message, chosen ciphertext etc. Therefore, broadcast attacks can be prevented by using one of the CCA2 conversions that have been proposed for these other types of attacks.

3.1 Unsuitable Conversions

Padding schemes like the well-known Optimal Asymmetric Encryption Padding (OAEP) by Bellare and Rogaway [1] are unsuitable for the McEliece/Niederreiter cryptosystems since they do not prevent reaction attacks: By randomly flipping bits and observing the reaction of the receiver, an attacker can recover the cleartext, and apply the conversion to reveal the message.

There are (at least) two generic conversion, proposed by Pointcheval [16] and by Fujisaki and Okamoto [8], that work with the McEliece and Niederreiter cryptosystems. However, they have the disadvantage of adding a large amount of redundancy to the ciphertexts. In both cases, the general idea is the following: Instead of encrypting the message with the (asymmetric) cryptosystem, say McEliece, it is encrypted with a symmetric system, and the corresponding key is encrypted with McEliece. Both outputs are appended to form the ciphertext. Since the output block size of McEliece and Niederreiter is large, a lot of redundancy is thereby added, which decreases the efficiency.

3.2 Kobara-Imai Conversion

More suitable is the conversion by Kobara and Imai [10]. This conversion was proposed for the McEliece cryptosystem, and for large messages it manages to reduce the redundancy of the ciphertext even below that of the unconverted cryptosystem. This conversion can also be applied to the Niederreiter cryptosystem. It can not be applied to the HyMES cryptosystem, since it uses a similar technique to encode part of the message into the error vector, hence applying the Kobara-Imai-conversion to the McEliece cryptosystem will achieve a similar efficiency improvement as the HyMES scheme. For the sake of completeness, we will briefly describe this conversion here. A more detailed description can be found in [2]. The resulting cryptosystem is a CCA2-secure variant of Niederreiter, which allows to implement secure and efficient cryptographic applications.

This conversion consists of two modifications. Firstly, it introduces randomness into the message, thereby rendering the output indistinguishable from a random ciphertext. This prevents attacks that rely on the relation of ciphertexts and/or cleartexts, e.g. message-resend, related-message, or broadcast attacks. Secondly, both the message vector m as well as the error vector e are computed from the message. This prevents reaction attacks, since a modified error vector results in a different cleartext, which can be detected by the honest user.

The pseudo code of the conversion can be found in the appendix.

4 Implementation

We have implemented the broadcast attack described above in Java. Target was the Niederreiter cryptosystem in the FlexiProvider package [6]. Table 2 shows the runtime for different parameters on an Intel i5-2500 CPU using one core. Note that our attack is not tweaked for performance, so we expect that this time can be improved further.

Table 2. Runtime of our algorithm for different parameters sets for the Niederreiter encryption scheme

Parameters (n, k, t)	Security level against ISD	Number of recipients	Runtime in sec
$(1024, 764, 26)$	57	4	6
$(1024, 524, 50)$	58	3	6
$(2048, 948, 100)$	97	2	35
$(2048, 1498, 50)$	101	4	50
$(4096, 3136, 80)$	174	5	460
$(4096, 2896, 100)$	184	4	430
$(4096, 2716, 115)$	188	3	352

Note that the runtimes increase cubic in n. This is expected, since the main work of the attack is to solve a system of linear equations, the complexity of which is in $\mathcal{O}(n^3)$.

5 Conclusion and Outlook

In this paper we have shown how a broadcast attack can be mounted against the Niederreiter and HyMES cryptosystem. We have calculated the number of recipients that are required in order to run our attack. Even though this number is usually very small, it is possible that an attacker intercepts only a smaller number of messages. We have shown that is it still possible to use this information to run a broadcast attack using Information Set Decoding, and the complexity of this attack decreases exponentially with the number of messages intercepted.

Our results have been tested experimentally, and our Java implementation was able to recover the secret message in < 20 seconds. The tests were run on an Intel Core i5 3.3 GHz CPU (one core), using the Niederreiter parameters $(n, t) = (2048, 50)$.

Finally, we showed that this type of attack can be prevented by applying a conversion on the message, e.g. Kobara-Imai's γ conversion.

As further work we propose to analyze if a structured code, e.g. a quasi-cyclic or quasi-dyadic code, is more vulnerable to this attack, resulting in a smaller number of required messages.

References

1. Bellare, M., Rogaway, P.: Optimal Asymmetric Encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995)
2. Bernstein, D.J., Buchmann, J., Dahmen, E.: Post-Quantum Cryptography. Springer (2008)
3. Bernstein, D.J., Lange, T., Peters, C.: Attacking and Defending the McEliece Cryptosystem. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 31–46. Springer, Heidelberg (2008)
4. Bernstein, D.J., Lange, T., Peters, C.: Wild McEliece. Cryptology ePrint Archive, Report 2010/410 (2010), http://eprint.iacr.org/
5. Biswas, B.: Implementational aspects of code-based cryptography. PhD thesis, École Polytechnique, Paris, France (2010)
6. Buchmann, J.: FlexiProvider. Developed by the Theoretical Computer Science Research Group of Prof. Dr. Johannes Buchmann at the Departement of Computer Science at Technische Universität Darmstadt, Germany, http://www.flexiprovider.de/
7. Faugère, J.-C., Otmani, A., Perret, L., Tillich, J.-P.: A Distinguisher for High Rate McEliece Cryptosystems. eprint Report 2010/331 (2010)
8. Fujisaki, E., Okamoto, T.: Secure Integration of Asymmetric and Symmetric Encryption Schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999)
9. Håstad, J.: Solving simultaneous modular equations of low degree. SIAM J. Comput. 17(2), 336–341 (1988)
10. Kobara, K., Imai, H.: Semantically Secure McEliece Public-Key Cryptosystems - Conversions for McEliece PKC. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 19–35. Springer, Heidelberg (2001)
11. Li, Y.X., Deng, R.H., Wang, X.M.: The equivalence of McEliece's and niederreiter's public-key cryptosystems. IEEE Trans. Inform. Theory 40, 271–273 (1994)
12. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. DNS Progress Report, 114–116 (1978)
13. Niebuhr, R., Cayrel, P.-L., Bulygin, S., Buchmann, J.: On lower bounds for Information Set Decoding over $\mathbb{F}_q$. In: SCC 2010, RHUL, London, UK (2010)
14. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. Problems of Control and Information Theory 15(2), 159–166 (1986)
15. Plantard, T., Susilo, W.: Broadcast Attacks against Lattice-Based Cryptosystems. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 456–472. Springer, Heidelberg (2009)
16. Pointcheval, D.: Chosen-Ciphertext Security for Any One-Way Cryptosystem. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 129–146. Springer, Heidelberg (2000)
17. Sendrier, N.: On the security of the McEliece public-key cryptosystem. In: Blaum, M., Farrell, P.G., van Tilborg, H. (eds.) Information, Coding and Mathematics, pp. 141–163. Kluwer (2002); Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday
18. Deng, Y., Pan, Y.: A broadcast attack against ntru using ding's algorithm. Cryptology ePrint Archive, Report 2010/598 (2010), http://eprint.iacr.org/

Appendix: Pseudo Code for the Kobara-Imai Conversion

Algorithm 6. Kobara-Imai's γ conversion, modified for the Niederreiter cryptosystem.

Notation for Algorithm 6:
$\mathbf{Enc^N}$: The Niederreiter encryption
$\mathbf{Dec^N}$: The Niederreiter decryption

Additional system parameters: The length of the random seed len_s, and a constant *const*.

Encryption Enc$^\gamma$
INPUT: Message $m \in \mathbb{F}_q^*$
OUTPUT: Ciphertext $c \in \mathbb{F}_q^t$, where $t = r + \text{len}(m) + \text{len}(c) + \text{len}(s) - l$

Generate a random seed s of length len_s
$y_1 \leftarrow \text{PRG}(s) \oplus (m|const)$
$y_2 \leftarrow s \oplus h(y_1)$

$(y_4|y_3) \leftarrow (y_2|y_1)$, such that
$\text{len}(y_3) = l$
$\text{len}(y_4) = \text{len}(m) + \text{len}(c) + \text{len}(s) - l$

$z \leftarrow \varphi(y_3)$
$c \leftarrow y_4|\mathbf{Enc^N}(z)$

Return c

Decryption Dec$^\gamma$
INPUT: Ciphertext $c \in \mathbb{F}_q^t$, where $t = r + \text{len}(m) + \text{len}(c) + \text{len}(s) - l$
OUTPUT: Message $m \in \mathbb{F}_q^*$

$y_4 \leftarrow \text{MSB}_{\text{len}(c)-n}(c)$
$z \leftarrow \mathbf{Dec^N}(\text{LSB}_n(c))$
$y_3 \leftarrow \varphi^{-1}(z)$
$(y_2|y_1) \leftarrow (y_4|y_3)$
$r \leftarrow y_2 \oplus h(y_1)$
$(m|const') \leftarrow y_1 \oplus \text{PRG}(s)$

if const'=const **then**
  Return m
else
  Reject c
end if

On the Security of Hummingbird-2 against Side Channel Cube Attacks

Xinxin Fan and Guang Gong

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
{x5fan,ggong}@uwaterloo.ca

Abstract. Hummingbird-2 is a recently proposed ultra-lightweight cryptographic algorithm targeted for resource-constrained devices like RFID tags, smart cards, and wireless sensor nodes. In this paper, we address the security of the Hummingbird-2 cipher against side channel cube attacks under the single-bit-leakage model. To this end, we describe an efficient term-by-term quadraticity test for extracting simple quadratic equations besides linear ones, obtainable from the original cube attack proposed by Dinur and Shamir at EUROCRYPT 2009. Moreover, we accelerate the implementation of the proposed term-by-term quadraticity test by fully exploiting the power of a Graphic Processing Unit (GPU). Our experimental results show that using a single bit of the internal state during the initialization process of the Hummingbird-2 cipher we can recover the 48 out of 128 key bits of the Hummingbird-2 with a data complexity of about 2^{18} chosen plaintexts.

Keywords: Algebraic cryptanalysis, side channel attacks, cube attacks, quadraticity test, Hummingbird-2.

1 Introduction

A cryptosystem can be seen as a set of Boolean functions, each of which is represented as a polynomial modulo 2 in what is known as its Algebraic Normal Form (ANF). The cube attack, announced by Dinur and Shamir in 2008 [7] and published at EUROCRYPT 2009 [8], is a generic key-recovery attack that may be applied to any cryptosystem, provided that the adversary can obtain a bit of information that can be represented by a low-degree decomposition multivariate polynomial in ANF of the secret and public variables of the target cryptosystem. An interesting feature of the cube attack is that it only requires a black-box access to a cryptosystem, that is, the internal structure of the target cryptographic primitive is unknown. The basic idea of the cube attack can also be found in some previous works such as the Algebraic IV Differential Attack (AIDA) proposed by Vielhaber [17,18] and the Higher Order Differential Attack proposed by Lai [14] and Knudsen [13]. In practice, the cube attack has been successfully applied to the reduced-round variants of the stream ciphers Trivium [8,17] and Grain [3,10], and to the reduced version of the hash function MD6 [4].

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 18–29, 2012.

In a typical design of iterated block ciphers, the degree of the Boolean function representing the output bit increases exponentially with the number of rounds. As a result, under the standard attack model (i.e., the attacker only has access to the plaintext and ciphertext) the cube attack might become ineffective after a few rounds. However, a stronger attack model, namely the side channel attack model, is usually applicable to the practical implementations of symmetric-key and public-key cryptosystems, especially on various embedded devices. In the side channel attack model, an adversary is able to access some *limited* information leaked about the internal state of a cryptographic primitive by analyzing the power consumption or the electromagnetic radiations of the device under attack [15]. Based on this observation, Dinur and Shamir [9] proposed the side channel cube attack based on the combination of the original cube attack and the single-bit-leakage model. More specifically, in the side channel cube attack the adversary is assumed to have access to only one bit of information about the internal state of a block cipher after each round. Since the introduction of the side channel cube attack, it has been used to recover the secret key for a couple of (lightweight) block ciphers such as Serpent and AES [9], PRESENT [2,19,20], NOEKEON [1], and KATAN [5].

Hummingbird-2 [12] is a recently proposed lightweight cryptographic primitive for resource-constrained smart devices such as RFID tags, smart cards, wireless sensor nodes, etc. It has been demonstrated that Hummingbird-2 can be efficiently implemented across a wide range of embedded software and hardware platforms [12]. In this contribution, we investigate the security of the Hummingbird-2 cipher against the side channel cube attack under the single-bit-leakage model. To this end, we propose an efficient term-by-term quadraticity test that extracts simple quadratic equations from the original cube attack, and implement the proposed quadraticity test on a Nvidia graphic card in a massively parallel fashion. Our experimental results show that one can recover the first 48 key bits of Hummingbird-2 by using a single bit of the internal state during the initialization process of the Hummingbird-2 cipher. Moreover, our attack has a time complexity of 2^{80} and a data complexity of about 2^{18} chosen plaintexts.

The remainder of this paper is organized as follows. Section 2 gives a brief review about the cube attack, followed by the description of the initialization procedure of the Hummingbird-2 cryptographic algorithm in Section 3. In Section 4, we present the term-by-term linearity and quadraticity tests in detail. Section 5 describes the massively parallel implementation the side channel cube attack against the Hummingbird-2 cipher and reports our experimental results. Finally, Section 6 concludes this paper.

2 A Review on the Cube Attack

The cube attack is a generic attack that can be applied to block ciphers, stream ciphers as well as keyed hash functions without necessarily knowing the internal structure of a cryptographic primitive. In the cube attack, a cryptographic primitive is viewed as a set of multivariate polynomials $p(v_1, \cdots, v_m, k_1, \cdots, k_n)$

over $\mathbb{F}_2$, each of them mapping m public variables v_i (i.e., plaintext bits in block ciphers and keyed hash functions or initial values in stream ciphers) and n secret variables k_i (i.e., key bits) to one of the ciphertext bits. The cube attack works as long as at least one output bit of the ciphertext can be represented by a low-degree multivariate polynomial of the public and secret variables. Let $I = \{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, m\}$ be a subset of the public variable indices and $t_I = x_{i_1} x_{i_2} \cdots x_{i_k}$ be a monomial term. Then the polynomial p, which is called a *master* polynomial, is decomposed as follows:

$$p(v_1, \cdots, v_m, k_1, \cdots, k_n) = t_I \cdot p_{S(I)} + q(v_1, \cdots, v_m, k_1, \cdots, k_n),$$

where t_I is called a *cube* that contains only a subset of public variables and $p_{S(I)}$ is called the *superpoly* of t_I in p. Note that the superpoly of I in p does not contain any common variables with t_I and each monomial in q does not contain at least one variable from I. A term t_I is called a *L-maxterm* if $p_{S(I)}$ is linear or *Q-maxterm* if $p_{S(I)}$ is quadratic, which is not a constant.

The main observation of the cube attack is that the symbolic sum over $\mathbb{F}_2$ of all evaluations of p by assigning all the possible combinations of 0/1 values to the public variables v_i's with $i \in I$ and fixing the value of all the remaining v_i's with $i \notin I$ is exactly $p_{S(I)}$ modulo 2, the superpoly of t_I in p. This observation enables an adversary to derive a linear equation over secret variables for any maxterm. After collecting enough linear equations, the adversary can recover the value of secret variables by using the Gaussian elimination method.

The cube attack consists of a *pre-processing* (offline) and an *online* phase. While the pre-processing phase aims to find monomials t_I's that lead to linear superpolys, the online phase solves the linear equations obtained from the pre-processing phase to recover the secret key. In order to find a sufficient number of maxterms t_I's during the pre-processing phase, the attacker first fixes the dimension of the cube as well as the value of public variables that are not in t_I. The attacker then performs a probabilistic linearity test (see Section 4 for details) on $p_{S(I)}$ over the secret variables $k_i \in \{k_1, \ldots, k_n\}$. If $p_{S(I)}$ passes the linearity test, with high probability $p_{S(I)}$ is linear over the secret variables and t_I is a maxterm. Since the maxterms are not key dependent, the pre-processing phase is performed once for a given cryptographic primitive. After sufficiently many linearly independent superpolys have been found, the online phase is conducted by evaluating the superpolys (i.e., summing up the master polynomial p over the same set of maxterms t_I's that are obtained in the pre-processing phase). Then the attacker can recover the secret key by solving the system of linear equations.

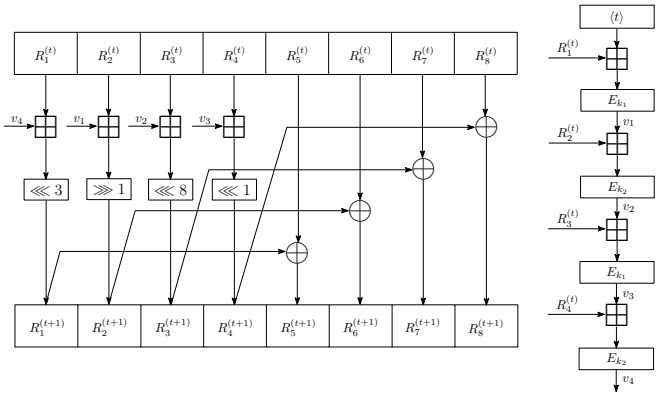
3 The Initialization of Hummingbird-2

Hummingbird-2 is a security enhanced version of its predecessor Hummingbird-1 [11], in response to the cryptanalysis work in [16]. The design of Hummingbird-2 adopts the same *hybrid* structure of block cipher and stream cipher as the Hummingbird-1 with 16-bit block size, 128-bit key size, and 128-bit internal state. To launch the cube attack, we solely focus on the initialization process of the

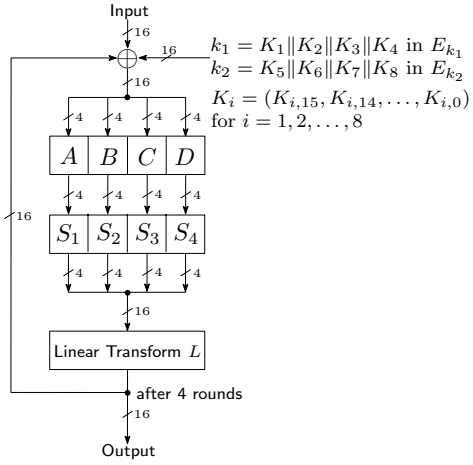
Hummingbird-2 as shown in Fig. 1(a), where $\boxplus$ denotes an addition modulo 2^{16}, $\oplus$ an exclusive-OR operation, and $\lll$ (or $\ggg$) a left (or right) circular shift operation, respectively. The initialization process consists of four 16-bit block ciphers E_{k_i} $(i = 1, 2)$ and eight 16-bit internal state registers $R_i^{(t)}$ $(i = 1, \ldots, 8$ and $t = 0, 1, \ldots)$. Initially, the register $R_i^{(0)}$ is set as follows:

$$R_i^{(0)} = \begin{cases} \mathsf{NONCE}_i & \text{for } i = 1, 2, 3, 4 \\ \mathsf{NONCE}_{i \bmod 4} & \text{for } i = 5, 6, 7, 8 \end{cases},$$

where NONCE_i $(i = 1, \ldots, 4)$ is the i-th 16-bit nonce. The 128-bit secret key K is divided into two 64-bit subkeys k_1 and k_2 which are used in the four block ciphers E_{k_i} $(i = 1, 2)$, respectively.



(a) Initialization Process



(b) 16-bit Block Cipher $E_{k_i}, i = 1, 2$

Fig. 1. Hummingbird-2 Initialization and Building Blocks

The Hummingbird-2 initialization employs four identical block ciphers $E_{k_i}(\cdot)$ $(i = 1, 2)$ in a consecutive manner, each of which is a typical substitution-permutation (SP) network with 16-bit block size and 64-bit key as shown in Fig. 1(b). The block cipher consists of four rounds, each of which is comprised of a key mixing step, a substitution layer, and a permutation layer. The key mixing step is implemented using a simple exclusive-OR operation, whereas the substitution layer is composed of four S-boxes $4, 5, 6$ and 7 of the Serpent block cipher [6] with 4-bit inputs and 4-bit outputs. The permutation layer in the 16-bit block cipher is given by the linear transform $L : \{0,1\}^{16} \to \{0,1\}^{16}$ defined as follows:

$$L(x) = x \oplus (x \lll 6) \oplus (x \lll 10),$$

where $x = (x_{15}, x_{14}, \ldots, x_0)$ is a 16-bit data block. The 64-bit subkeys k_i $(i = 1, 2)$ are split into four 16-bit round keys (see Figure 1(b)) that are used in the four rounds, respectively. The entire initialization process consists of four rounds and after each round the eight internal state registers are updated from $R_i^{(t)}$ to $R_i^{(t+1)}$ $(i = 1, \ldots, 8)$ in an unpredictable way based on their current states as well as the outputs of the 16-bit block ciphers. For more details about the Hummingbird-2 cipher, the interested reader is referred to [12].

For applying the side channel cube attack to the Hummingbird-2 initialization process under the single-bit leakage model, we assume that there is a single bit leakage after the third round of the first 16-bit block cipher E_{k_1}. This enables us to recover the first 48 bits of the secret key K in the Hummingbird-2. As an example of the attack, we provide the analysis results for the least significant bit of the internal state after the third round of E_{k_1} in this contribution.

4 Linearity and Quadraticity Tests

Let $\mathbb{F}_2^n$ be an n-dimensional vector space over the finite field $\mathbb{F}_2$ and $f(x_1, \ldots, x_n) : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function of n variables. One of the crucial steps in the cube attack is to test whether the Boolean function f is linear or quadratic, and if so, we need to find out the expression of f. More specifically, we need to consider the following two cases:

- A Boolean function f is linear in its inputs if it satisfies $f(x \oplus y) = f(x) \oplus f(y) \oplus f(0)$ for all $x, y \in \mathbb{F}_2^n$. Such a linear function has a form of $f(x_1, \ldots, x_n) = \bigoplus_{1 \le i \le n} a_i x_i \oplus a_0$, where $a_i \in \mathbb{F}_2$ and $a_0 = f(0)$.
- A Boolean function f is quadratic in its inputs if it satisfies $f(x \oplus y \oplus z) = f(x \oplus y) \oplus f(x \oplus z) \oplus f(y \oplus z) \oplus f(x) \oplus f(y) \oplus f(z) \oplus f(0)$ for all $x, y, z \in \mathbb{F}_2^n$. Such a quadratic function has a form of $f(x_1, \ldots, x_n) = \bigoplus_{1 \le i < j \le n} a_{ij} x_i x_j \oplus \bigoplus_{1 \le i \le n} a_i x_i + a_0$, where $a_{ij}, a_i \in \mathbb{F}_2$ and $a_0 = f(0)$.

In [21], Zhu *et al.* proposed an efficient *term-by-term* linearity test (see Algorithm 1 in Fig. 2) in which the Boolean function f needs to be evaluated $n+1+2 \cdot d_1 \cdot C_1$ times in order to discover the linear secret variables within a superpoly equation and test their linearity, where n is the number of secret variables, d_1 is the number

Algorithm 1 Term-by-Term Linearity Test [21]
$S_l \leftarrow$ an empty set
for $i \leftarrow 1$ **to** n **do**
  $x \leftarrow (0, 0, \ldots, 1, \ldots, 0)$ where only the i-th bit is 1
  if $f(x) \oplus f(0) = 1$ **then**
    Add i into the set S_l
  end if
end for
for each $j \in S_l$ **do**
  for $c \leftarrow 1$ **to** C_1 **do**
    Randomly choose an input value $y \in \mathbb{F}_2^n$
    $y' \leftarrow y$
    $y_j \leftarrow 0$ and $y'_j \leftarrow 1$
    if $f(y) = f(y')$ **then**
      Reject and halt
    end if
  end for
end for

Algorithm 2 Term-by-Term Quadraticity Test
$S_l \leftarrow$ an empty set
$S_q \leftarrow$ an empty set
Execute the term-by-term linearity test (see the Algorithm 1)
for $i \leftarrow 1$ **to** $n - 1$ **do** //Quadratic term discovery phase
  for $j \leftarrow i + 1$ **to** n **do**
    $x \leftarrow (0, \ldots, 1, \ldots, 1, \ldots, 0)$ where only the i-th and j-th bits are 1
    if $(f(x) \oplus f(0) = 1$ **and** $(i, j \in S_l$ **or** $i, j \notin S_l))$ **or**
      $(f(x) \oplus f(0) = 0$ **and** $(i \in S_l, j \notin S_l$ **or** $i \notin S_l, j \in S_l)))$ **then**
        Add (i, j) into the set S_q
    end if
  end for
end for
for each $(i, j) \in S_q$ **do** //Quadraticity testing phase
  for $c \leftarrow 1$ **to** C_2 **do**
    Randomly choose an input value $y^{(1)} \in \mathbb{F}_2^n$
    $y^{(2)} \leftarrow y^{(1)}, y^{(3)} \leftarrow y^{(1)}$, and $y^{(4)} \leftarrow y^{(1)}$
    $(y_i^{(1)}, y_j^{(1)}) \leftarrow (0, 0)$ and $(y_i^{(2)}, y_j^{(2)}) \leftarrow (0, 1)$
    $(y_i^{(3)}, y_j^{(3)}) \leftarrow (1, 0)$ and $(y_i^{(4)}, y_j^{(4)}) \leftarrow (1, 1)$
    if $f(y^{(1)}) \oplus f(y^{(2)}) = f(y^{(3)}) \oplus f(y^{(4)})$ **then**
      Reject and halt
    end if
  end for
end for

Fig. 2. The Term-by-Term Linearity and Quadraticity Tests

of linear terms, and C_1 is the total number of linearity tests. We generalize this approach to the quadratic case and obtain a faster term-by-term quadraticity test (see Algorithm 2 in Fig. 2). The basic idea is to first find all linear and quadratic terms in the superpoly equation, followed by a probabilistic linearity (and/or quadraticity) test for each individual linear (and/or quadratic) term. In Algorithm 2, we first define two sets S_l and S_q for storing the indices of linear and quadratic terms, respectively. In order to check whether a quadratic term k_ik_j belongs to a Boolean function f, we take $x = (0, \ldots, 1, \ldots, 1, \ldots, 0)$ with the i-th and j-th bits set to 1 and consider the following three cases:

- Case 1: $i, j \in S_l$. In this case, if k_ik_j is a term of f, the Boolean function f contains $k_ik_j \oplus k_i \oplus k_j$. Hence we obtain $f(x) \oplus f(0) = 1$.
- Case 2: $i, j \notin S_l$. In this case, if the quadratic term k_ik_j belongs to f, the condition $f(x) \oplus f(0) = 1$ still holds.
- Case 3: $i \in S_l, j \notin S_l$ or $i \notin S_l, j \in S_l$. In this case, if k_ik_j is a term of f, the Boolean function f contains $k_ik_j \oplus k_i$ or $k_ik_j \oplus k_j$. Therefore, we obtain $f(x) \oplus f(0) = 0$.

In this way, we are able to find out all the quadratic terms in the Boolean function f. The next step is to perform a probabilistic quadraticity test C_2 times for each pair of indices $(i, j) \in S_q$. To this end, we first randomly generate an n-bit vector $y^{(1)} \in \mathbb{F}_2^n$. Then we vary the i-th and j-th bits of $y^{(1)}$ and generate other three n-bit vectors $y^{(2)}, y^{(3)}$ and $y^{(4)}$ that are equal to $y^{(1)}$ except for the i-th and j-th bits. More specifically, we have $(y_i^{(1)}, y_j^{(1)}) = (0, 0), (y_i^{(2)}, y_j^{(2)}) = (0, 1), (y_i^{(3)}, y_j^{(3)}) = (1, 0)$ and $(y_i^{(4)}, y_j^{(4)}) = (1, 1)$. Finally, we test whether the equation $f(y^{(1)}) \oplus f(y^{(2)}) = f(y^{(3)}) \oplus f(y^{(4)})$ holds. If it does, k_ik_j is not a quadratic term in f. Essentially, the above process eliminates k_i from the Boolean function f and checks whether k_j is a linear term, which is clearly demonstrated in the following example.

Example 1. Assuming that a Boolean function $f(k_1, k_2, k_3) = k_3k_2 \oplus k_3k_1 \oplus k_2k_1$, we obtain $(3, 2), (3, 1), (2, 1) \in S_q$ after a quadratic term discovery phase in Algorithm 2. To test whether k_3k_2 is a quadratic term in f, we first set $k_3 = 0$ and 1, respectively, which gives us

$$f_0 = f(k_1, k_2, 0) = k_2k_1, \tag{1}$$

$$f_1 = f(k_1, k_2, 1) = k_2k_1 \oplus k_1 \oplus k_2. \tag{2}$$

Adding (1) and (2), we obtain $f_0 \oplus f_1 = k_1 \oplus k_2$. Therefore, the remaining task is to check whether k_2 is linear in $f_0 \oplus f_1$, which is exactly what we have done in the quadraticity testing phase of the Algorithm 2.

Using the term-by-term quadraticity test in Algorithm 2, the Boolean function f needs to be totally evaluated $(n + 1 + 2 \cdot d_1 \cdot C_1) + \left(\frac{n(n-1)}{2} + 4 \cdot d_2 \cdot C_2\right)$ times for a superpoly with d_1 linear terms and d_2 quadratic terms, where C_1 and C_2 are the number of linearity and quadraticity tests, respectively.

5 Side Channel Cube Attack on Hummingbird-2

As shown in Fig. 1(b), the 64-bit subkey k_1 in the block cipher E_{k_1} is divided into four 16-bit round keys $K_i = (K_{i,15}, \ldots, K_{i,0}), i = 1, 2, 3, 4$. At the i-th round the round key K_i is exclusive-ORed with the internal state of the block cipher E_{k_1}. Hence, after the third round, the internal state of the block cipher E_{k_1} contains the information about the round keys K_1, K_2 and K_3. In order to find the maxterms from a master polynomial associated with the least significant bit (LSB)[1] of the internal state after the third round, we exhaustively test all possible cube sizes ranging from 1 to 16 (Recall that in the first round of the Hummingbird-2 initialization the input to the first block cipher E_{k_1} is the 16-bit nonce NONCE_1) with the help of a modern Graphic Processing Unit (GPU) (i.e., a GeForce GTX 285 graphics card) from Nvidia.

5.1 GPU Assisted Parallel Cube Attack

A general purpose GPU contains a large number of processor cores that run at frequencies that are mostly lower than CPUs. When compared to a CPU, a GPU provides a much larger computational power for specific parallel applications, because of the large number of cores. The GTX 285 GPU is based on the Tesla architecture, which features an array of multiprocessors (30 for the GTX 285) that each contains 8 scalar processors. Each scalar processor is able to perform a 32-bit operation each cycle. The computational power of the GPU is exploited by running many hardware threads in parallel: Each core handles threads in groups of 32 called *warps* in a single-instruction-multiple-threads (SIMT) approach. The Compute Unified Device Architecture (CUDA) programming framework developed by Nvidia provides a C-like language to program Nvidia GPUs.

Regarding to the side channel cube attack on Hummingbird-2, we significantly accelerate the evaluation of the master polynomial $p(v_1, \cdots, v_m, k_1, \cdots, k_n)$ by launching 2^κ (κ is the size of a cube) threads simultaneously, each of which calculates the value of the master polynomial for one of 2^κ different 0/1 combinations of a subset of public variables $(v_{i_1}, v_{i_2}, \ldots, v_{i_\kappa})$. After we obtain 2^κ values of the master polynomial, a parallel reduction process is conducted to exclusive-OR the 2^κ values, yielding the value of the superpoly $p_{S(I)}$. The parallel implementation of the evaluation of the superpoly $p_{S(I)}$ on the GTX 285 is illustrated in Fig. 3. Based on the parallel superpoly evaluation in Fig. 3, the proposed term-by-term quadraticity test can be efficiently implemented in a massively parallel fashion on the target GPU.

5.2 Experimental Results

After running the faster term-by-term quadraticity test (see Algorithm 2 in Section 4) on a single PC (with a GPU) for a few days, we have been able to

[1] The cube attack can also be applied to any other single bit of the internal state after the third round of the block cipher E_{k_1} in a straightforward way.

Algorithm 3 Parallel Superpoly Evaluation on GPUs
Inputs: A master polynomial $p(v_1, \cdots, v_m, k_1, \cdots, k_n) = t_I \cdot p_{S(I)} + q(v_1, \cdots, v_m, k_1, \cdots, k_n)$
Cube indices : $\{i_1, i_2, \ldots, i_\kappa\}$ and a certain key $k = (k_1, \cdots, k_n)$
Output: The value of the superpoly $p_{S(I)}$
parallel for $l \leftarrow 0$ **to** $2^\kappa - 1$ **do** // parallel evaluation
$l = (l_0, l_1, \ldots, l_\kappa)$
$(v_{i_1}, v_{i_2} \cdots, v_{i_\kappa}) \leftarrow (l_0, l_1, \ldots, l_\kappa)$ and $v_j = 0$ for $j \in \{1, 2, \ldots, m\} \setminus \{i_1, i_2, \ldots, i_\kappa\}$
Compute $p_l(v_1, \cdots, v_m, k_1, \cdots, k_n)$ // each thread computes one value
end parallel for
$p_{S(I)} \leftarrow \sum_{l=0}^{2^\kappa - 1} p_l(v_1, \cdots, v_m, k_1, \cdots, k_n)$ // parallel reduction using shared memory

Fig. 3. Parallel Evaluation of the Superpoly $p_{S(I)}$ on GPUs

Table 1. The Cube Indices and Superpoly Equations for the Hummingbird-2 Initialization from the Least Significant Bit Leakage after the Third Round of the First E_{k_1}

Cube Indices	Cube Size	Linear Superpoly Equation
$\{11, 10, 9, 8, 7, 6, 5, 4, 3, 2\}$	10	$K_{1,0} + K_{1,1} + 1$
$\{15, 14, 13, 12, 7, 6, 5, 4, 3, 0\}$	10	$K_{1,1}$
$\{11, 10, 9, 8, 7, 6, 5, 4, 3, 0\}$	10	$K_{1,2} + 1$
$\{11, 10, 9, 8, 7, 6, 5, 4, 2, 0\}$	10	$K_{1,3}$
$\{11, 10, 9, 8, 7, 3, 2, 1, 0\}$	9	$K_{1,4}$
$\{15, 14, 13, 12, 11, 10, 9, 8, 6, 4\}$	10	$K_{1,5}$
$\{15, 14, 13, 12, 4, 3, 2, 1, 0\}$	9	$K_{1,5} + K_{1,6}$
$\{11, 10, 9, 8, 4, 3, 2, 1, 0\}$	9	$K_{1,7}$
$\{15, 14, 13, 12, 10, 9, 3, 2, 1, 0\}$	10	$K_{1,8}$
$\{11, 8, 7, 6, 5, 4, 3, 2, 1, 0\}$	10	$K_{1,9}$
$\{15, 14, 13, 12, 9, 8, 3, 2, 1, 0\}$	10	$K_{1,10}$
$\{15, 14, 13, 12, 9, 8, 7, 6, 5, 4\}$	10	$K_{1,11}$
$\{14, 13, 7, 6, 5, 4, 3, 2, 1, 0\}$	10	$K_{1,12}$
$\{15, 12, 7, 6, 5, 4, 3, 2, 1, 0\}$	10	$K_{1,13} + K_{1,14}$
$\{15, 12, 11, 10, 9, 8, 7, 6, 5, 4\}$	10	$K_{1,14}$
$\{15, 12, 11, 10, 9, 8, 7, 6, 5, 4\}$	10	$K_{1,15} + 1$
$\{14, 13, 12, 10, 9, 8, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,0} + K_{2,5} + K_{2,6} + K_{2,7} + K_{2,9} + K_{2,11} + K_{2,13}$
$\{14, 13, 12, 11, 9, 8, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,3} + K_{2,5} + K_{2,6} + K_{2,7} + K_{2,10} + K_{2,11} + K_{2,13} + 1$
$\{14, 13, 12, 11, 10, 8, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,2} + K_{2,3} + K_{2,6} + K_{2,7} + K_{2,8} + K_{2,10} +$ $K_{2,11} + K_{2,13} + K_{2,14} + K_{2,15} + 1$
$\{14, 13, 12, 11, 10, 9, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,0} + K_{2,2} + K_{2,3} + K_{2,5} + K_{2,8} +$ $K_{2,9} + K_{2,10} + K_{2,14} + K_{2,15}$
$\{14, 13, 12, 11, 10, 9, 8, 7, 6, 4, 2, 1, 0\}$	13	$K_{2,2} + K_{2,3} + K_{2,4} + K_{2,5} + K_{2,7} +$ $K_{2,9} + K_{2,10} + K_{2,11} + K_{2,13} + K_{2,14}$
$\{14, 13, 12, 11, 10, 9, 8, 7, 6, 4, 3, 1, 0\}$	13	$K_{2,5} + K_{2,8} + K_{2,11} + K_{2,14}$
$\{14, 13, 12, 11, 10, 9, 8, 7, 6, 4, 3, 2, 0\}$	13	$K_{2,0} + K_{2,10} + K_{2,12}$
$\{15, 13, 12, 10, 9, 8, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,5} + K_{2,7} + K_{2,9} + K_{2,11} + K_{2,13} + 1$
$\{15, 13, 12, 11, 9, 8, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,0} + K_{2,1} + K_{2,3} + K_{2,4} + K_{2,5} + K_{2,10} + K_{2,11} + K_{2,13}$
$\{15, 13, 12, 11, 10, 8, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,1} + K_{2,2} + K_{2,3} + K_{2,4} + K_{2,6} + K_{2,8} +$ $K_{2,10} + K_{2,11} + K_{2,13} + K_{2,14} + K_{2,15}$
$\{15, 13, 12, 11, 10, 9, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,1} + K_{2,2} + K_{2,3} + K_{2,4} + K_{2,5} + K_{2,6} +$ $K_{2,8} + K_{2,9} + K_{2,10} + K_{2,14} + K_{2,15}$
$\{15, 13, 12, 11, 10, 9, 8, 7, 6, 4, 2, 1, 0\}$	13	$K_{2,1} + K_{2,2} + K_{2,3} + K_{2,4} + K_{2,5} +$ $K_{2,10} + K_{2,12} + K_{2,13} + K_{2,14} + K_{2,15} + 1$
$\{15, 13, 12, 11, 10, 9, 8, 7, 6, 4, 3, 1, 0\}$	13	$K_{2,1} + K_{2,2} + K_{2,5} + K_{2,6} +$ $K_{2,7} + K_{2,9} + K_{2,12} + K_{2,14} + K_{2,15} + 1$
$\{15, 13, 12, 11, 10, 9, 8, 7, 6, 4, 3, 2, 0\}$	13	$K_{2,0} + K_{2,3} + K_{2,5} + K_{2,6} + K_{2,9} + K_{2,10} + K_{2,11} + 1$
$\{15, 14, 12, 11, 9, 8, 7, 6, 4, 3, 2, 1, 0\}$	13	$K_{2,0} + K_{2,1} + K_{2,3} + K_{2,4} + K_{2,6} + K_{2,7} + K_{2,9}$
$\{15, 14, 12, 11, 10, 9, 8, 7, 6, 4, 2, 1, 0\}$	13	$K_{2,1} + K_{2,2} + K_{2,3} + K_{2,8} + K_{2,11} + K_{2,13} + K_{2,15}$

find tens of linear and quadratic superpoly equations using different cube sizes. For the linear superpolys, we use the Gaussian elimination to remove the linear dependent equations and obtain 32 linear independent equations with 32 key variables as a result. Table 1 lists the indices of the public variables in the maxterms and the corresponding linear superpoly equations. For the quadratic superpolys, we find that they are all redundant and cannot provide more information about the secret key bits. As shown in Table 1, we have 3 maxterms of size 9, 13 maxterms of size 10, and 16 maxterms of size 13. Therefore, in order to recover the first 32 key bits of the secret key, the total number of chosen plaintexts (i.e., the nonce NONCE_1) at the online phase of the cube attack is $3\times 2^9+13\times 2^{10}+16\times 2^{13}\approx 2^{17.155}$. After obtaining the first 32 key bits K_1 and K_2, we can use those key bits to significantly simplify the master polynomial associated with the LSB of the internal state of the block cipher E_{k_1} after the third round. As an example, we assume that $K_1 = \mathsf{0x35df}$ and $K_2 = \mathsf{0xac2b}$. Using the Algorithm 2, we find 12 linear independent equations (see Table 2) with 12 secret key variables $K_{3,4}, K_{3,5}, \ldots, K_{3,15}$, which enable us to solve those key variables at the online phase with $2^4 + 2 \times 2^5 + 6 \times 2^6 + 3 \times 2^7 \approx 2^{9.755}$ chosen plaintexts. The remaining four secret key bits $K_{3,0}, K_{3,1}, K_{3,2}$ and $K_{3,3}$ can be obtained by conducting an exhaustive search. Moreover, one can also obtain the relations among those four key bits by solving the quadratic equations in Table 2 with another $2 \times 2^5 = 64$ chosen plaintexts. Consequently, the total time complexity to find the correct 128-bit secret key of the Hummingbird-2 has been reduced to 2^{80}.

Table 2. The Cube Indices and Superpoly Equations for the Hummingbird-2 Initialization from the Least Significant Bit Leakage after the Third Round of the First E_{k_1} (K_1 and K_2 are known)

Cube Indices	Cube Size	Superpoly Equation
$\{7,6,5,4\}$	4	$K_{3,4}+K_{3,6}+K_{3,7}+K_{3,8}+K_{3,10}+K_{3,11}+K_{3,12}+K_{3,14}+K_{3,15}+1$
$\{7,6,5,4,0\}$	5	$K_{3,4}+1$
$\{12,10,7,4,0\}$	5	$K_{3,12}+1$
$\{11,9,5,4,3,0\}$	6	$K_{3,10}$
$\{11,10,9,8,6,4\}$	6	$K_{3,13}$
$\{7,6,5,4,3,0\}$	6	$K_{3,6}+K_{3,7}+K_{3,12}$
$\{6,4,3,2,1,0\}$	6	$K_{3,4}+K_{3,9}+K_{3,10}+K_{3,11}+1$
$\{11,10,9,6,4,0\}$	6	$K_{3,4}+K_{3,5}+K_{3,8}+K_{3,10}+K_{3,15}+1$
$\{9,8,3,2,1,0\}$	6	$K_{3,5}+K_{3,7}+K_{3,8}+K_{3,11}+K_{3,14}+K_{3,15}+1$
$\{8,6,5,3,2,1,0\}$	7	$K_{3,6}+K_{3,7}+K_{3,8}+K_{3,9}$
$\{8,7,6,3,2,1,0\}$	7	$K_{3,6}+K_{3,7}+K_{3,8}+K_{3,10}+K_{3,11}+1$
$\{8,7,6,5,4,2,1\}$	7	$K_{3,5}+K_{3,6}+K_{3,8}+K_{3,9}+K_{3,10}+K_{3,13}+K_{3,14}+K_{3,15}$
$\{10,8,7,6,4\}$	5	$K_{3,7}+K_{3,8}+K_{3,9}+K_{3,11}+K_{3,12}+K_{3,13}+K_{3,14}+$ $K_{3,8}K_{3,11}+K_{3,9}K_{3,11}+K_{3,4}K_{3,5}+K_{3,5}K_{3,6}+K_{3,2}K_{3,3}$
$\{9,8,6,5,2\}$	5	$K_{3,4}+K_{3,7}+K_{3,9}+K_{3,10}+K_{3,12}+K_{3,9}K_{3,11}+$ $K_{3,10}K_{3,11}+K_{3,5}K_{3,7}+K_{3,4}K_{3,5}+K_{3,1}K_{3,2}+K_{3,0}K_{3,2}$

6 Conclusions

In this contribution we investigate the security of the Hummingbird-2 against the side channel cube attacks under the single-bit-leakage model. Our experimental results show that one can recover the first 48 bits of the secret key in the Hummingbird-2, by taking advantage of a single bit information leakage from the internal state after the third round of the first block cipher E_{k_1}. The data complexity of the proposed attack is around 2^{18}. Moreover, an efficient term-by-term quadraticity test is also proposed. Finally, we would like to point out that in order to launch the side channel cube attack against the Hummingbird-2 an attacker needs to acquire the exact value of the least significant bit after the third round of the block cipher E_{k_1}, which is, if not impossible, quite difficult and expensive in practice according to the current manufacturing technology of embedded systems. Therefore, the proposed attack is only of a theoretical interest at the moment and does not directly jeopardize the security of the Hummingbird-2 implementations in practice.

Acknowledgment. This work is supported by an NSERC Strategic Project Grant and the Ontario Research Fund Research Excellence (ORF-RE) program.

References

1. Abdul-Latip, S.F., Reyhanitabar, M.R., Susilo, W., Seberry, J.: On the Security of NOEKEON against Side Channel Cube Attacks. In: Kwak, J., Deng, R.H., Won, Y., Wang, G. (eds.) ISPEC 2010. LNCS, vol. 6047, pp. 45–55. Springer, Heidelberg (2010)
2. Abdul-Latip, S.F., Reyhanitabar, M., Susilo, W., Seberry, J.: Extended Cubes: Enhancing the Cube Attack by Extracting Low-Degree Non-Linear Equations. In: The 6th ACM Symposium on Information, Computer and Communications Security - ASIACCS 2011, pp. 296–305. ACM Press (2011)
3. Aumasson, J.-P., Dinur, I., Henzen, L., Meier, W., Shamir, A.: Efficient FPGA Implementations of High-Dimensional Cube Tester on the Stream Cipher Grain-128. In: The 4th International Workshop on Special-purpose Hardware for Attacking Cryptographic Systems - SHARCS 2009 (2009), http://www.131002.net/data/papers/ADHMS09.pdf
4. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009)
5. Bard, G.V., Courtois, N.T., Nakahara Jr., J., Sepehrdad, P., Zhang, B.: Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 176–196. Springer, Heidelberg (2010)
6. Anderson, R., Biham, E., Knudsen, L.R.: Serpent: A Proposal for the Advanced Encryption Standard (1999), http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf
7. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials, Cryptology ePrint Archive, Report 2008/385 (2008), http://eprint.iacr.org/2008/385

8. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
9. Dinur, I., Shamir, A.: Side Channel Cube Attacks on Block Ciphers, Cryptology ePrint Archive, Report 2009/127 (2009), http://eprint.iacr.org/2009/127
10. Dinur, I., Shamir, A.: Breaking Grain-128 with Dynamic Cube Attacks. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 167–187. Springer, Heidelberg (2011)
11. Engels, D., Fan, X., Gong, G., Hu, H., Smith, E.M.: Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) FC 2010 Workshops. LNCS, vol. 6054, pp. 3–18. Springer, Heidelberg (2010)
12. Engels, D., Saarinen, M.-J.O., Schweitzer, P., Smith, E.M.: The Hummingbird Lightweight Authenticated Encryption Algorithm. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 19–31. Springer, Heidelberg (2012)
13. Knudsen, L.R.: Truncated and High Order Differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
14. Lai, X.: Higher Order Derivatives and Differential Cryptanalysis. In: Communications and Cryptography: Two Sides of One Tapestry, pp. 227–233. Kluwer Academic Publishers (1994)
15. Le, T.-H., Canovas, C., Clédière, J.: An Overview of Side Channel Analysis Attacks. In: The 2008 ACM Symposium on Information, Computer and Communications Security - ASIACCS 2008, pp. 33–43. ACM Press (2008)
16. Saarinen, M.-J.O.: Cryptanalysis of Hummingbird-1. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 328–341. Springer, Heidelberg (2011)
17. Vielhaber, M.: Breaking ONE.FIVIUM by AIDA – an Algebraic IV Differential Attack, Cryptology ePrint Archive, Report 2007/413 (2007), http://eprint.iacr.org/2007/413
18. Vielhaber, M.: AIDA Breaks BIVIUM (A&B) in 1 Minute Dual Core CPU Time, Cryptology ePrint Archive, Report 2009/402 (2009), http://eprint.iacr.org/2009/402
19. Yang, L., Wang, M., Qiao, S.: Side Channel Cube Attack on PRESENT. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 379–391. Springer, Heidelberg (2009)
20. Zhao, X., Wang, T., Guo, S.: Improved Side Channel Cube Attacks on PRESENT, Cryptology ePrint Archive, Report 2011/165 (2011), http://eprint.iacr.org/2011/165
21. Zhu, B., Yu, W., Wang, T.: A Practical Platform for Cube-Attack-Like Cryptanalyses, Cryptology ePrint Archive, Report 2010/644 (2010), http://eprint.iacr.org/2010/644

Full Lattice Basis Reduction on Graphics Cards

Timo Bartkewitz[1] and Tim Güneysu[2]

[1] Department of Computer Science,
Bonn-Rhine-Sieg University of Applied Sciences,
Grantham-Allee 20, 53757 Sankt Augustin, Germany
timo.bartkewitz@h-brs.de
[2] Horst Görtz Institute for IT Security,
Ruhr-University Bochum, Germany
gueneysu@crypto.rub.de

Abstract. Recent lattice enumeration GPU implementations are very useful to find shortest vectors within a given lattice but are also highly dependent on a lattice basis reduction that still runs on a CPU. Therefore we present an implementation of a full lattice basis reduction that makes exclusive use of GPUs to close this gap. Hence, we show that GPUs are, as well, suited to apply lattice basis reduction algorithms that were merely of theoretical interest so far due to their enormous computational effort. We modified and optimized these algorithms to fit the architecture of graphics cards, in particular we focused on Givens Rotations and the All-swap reduction method. Eventually, our GPU implementation achieved a significant speed-up for given lattice challenges compared to the NTL implementation running on an CPU of about 18, providing at least the same reduction quality.

Keywords: Lattice Basis Reduction, Givens Rotations, All-Swap Algorithm, Parallelization, Graphics Cards, CUDA.

1 Introduction

The lattice basis reduction is an important and interesting tool in linear algebra. Various applications concern the factorization of polynomials and integer numbers as well as solving knapsack, hidden number problems, and many more problems [14,17] – all enabled by finding a relatively short lattice basis (*Shortest Basis Problem* or *SBP*) and the shortest vector for a given lattice (*Shortest Vector Problem* or *SVP*). In particular, the latter method could also be used to break instances of the RSA public-key cryptosystem [29]. Beside this factoring-based cryptosystem, there is the class of lattice-based cryptosystems that are assumed to be secure against attacks with quantum computers. Considering those crypto schemes, the lattice basis reduction can be applied to determine the hardness of respective lattice problems. Well-known lattice-based cryptosystems are NTRU [15], Merkle-Hellman [23], Ajtai-Dwork [2], Goldreich-Goldwasser-Halevi [10], but also recent cryptosystems by Regev [28], Peikert [27], as well as Applebaum-Cash-Peikert-Sahai [3], Stehl-Steinfeld-Tanaka-Xagawa [7], Brakerski-Goldwasser-Kalai [5] and

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 30–44, 2012.

SWIFFT [22]. Each scheme relies on the hardness of finding short vectors or closely related problems, respectively. Thus, the performance of lattice basis reduction is indeed essential for reasonable security estimations.

The first practical algorithm to find a short lattice basis vector was proposed in 1982 by Lenstra, Lenstra and Lovász, known as *LLL-algorithm* [21]. It provides polynomial runtime but involves a quality factor (*i.e.* a reduction parameter) which is exponentially dependent on the magnitude of the lattice. Additionally, the LLL-algorithm suffers from a further disadvantage: for correctness reasons, it applies exact integer arithmetic which has been shown to be inefficient. Hence, there were many efforts made to improve this algorithm concerning runtime behavior and the reachable reduction quality. Schnorr and Euchner therefore proposed a variant of the LLL-algorithm that applies floating point arithmetic and thus offers much better runtime results, namely the *Schnorr-Euchner-algorithm* (SE) [31]. Simultaneously, Schnorr and Euchner presented the hybrid BKZ-algorithm [31] that is most widely used for lattice basis reduction in practice today because it delivers the best reduction quality, especially with respect to a shortest vector. Crucially, it is also built on the LLL-algorithm and thus dependent on its performance. There are also recent LLL-algorithms to mention whose running time provably grows only quadratically [24].

Motivation: Concerning graphics cards, many results on lattice enumeration has been recently published but there are no results on the LLL, even less on the combination of both (BKZ) (see also Sec. 2). In contrast to other platforms that can potentially accommodate lattice basis reduction algorithms, graphics cards using the OpenCL [19] or CUDA [26] programming model provide a very promising platform for computationally intensive tasks. This platform allows to run a large number of parallel processors making it feasible to adopt parallel algorithms which were currently designated to be impractical due to their demand for huge computational resources (*i.e.* a large number of available processors), such as the *All-swap* LLL-algorithm [33]. It is still an open issue whether GPU-based lattice basis reduction algorithms are capable to compete with optimized CPU implementations like those provided by the number theory library (NTL) [32] for instance. Until today there are virtually no efforts made to port the lattice basis reduction to graphic cards. For sure, there are still obstacles (*e.g.* the restriction to double floating point precision) voting against graphics cards because accuracy is an important factor for lattice basis reduction in very high dimensions. But it is a matter of time when these problems are going to be solved by even more powerful devices and parallel algorithms then become highly important.

Our contribution: In this paper, we will adopt and improve parallel algorithms for lattice basis reduction to achieve optimal results on graphics cards. A major part of a LLL-based lattice basis reduction algorithm is the orthogonalization. A suggestion in [18] is to implement the orthogonalization step by using so-called Householder reflections [9]. Unlike, Givens rotations are assumed to be inefficient on graphics cards but we are not aware of any comparison nor a corresponding implementation on this platform. In this paper we thus present

an implementation of the Givens rotations that offers a better runtime performance (compared to the Householder reflections implementation by Kerr et al. [18] for equivalent parameters) which is achieved by the idea of merging matrix row computations. A further contribution is, mainly following the All-swap LLL-algorithm approach, to pick up the improvement made to the original LLL-algorithm of the so-called *deep insertion* technique [31] and apply it to our modified algorithm resulting in a method we call *sorted All-swap*. Ultimately, we present the first implementation of lattice basis reduction on graphics cards and provide further strategies and incentives for future works. We provide performance results for the lattice challenges published by TU Darmstadt [6] whose construction is based on [1], random lattices, and random lattices in Hermite normal form (HNF) using double floating point precision. Even though there are better methods than LLL or BKZ respectively, these lattices should serve as a common foundation for performance comparisons. For the given lattices, our implementation provides a significantly higher performance compared to the sequential Schnorr-Euchner-algorithm using Givens rotations contained in the NTL. Hence, we would also expect a better runtime when using much higher entries respectively dimensions which will be applicable with support for multi-precision floating point arithmetic in future generations of graphics cards.

Organization of the paper: This paper is organized as follows: Section 2 briefly introduces the previous work on parallel lattice basis reduction algorithms. In Section 3, we briefly present the mathematical background on lattices. Section 4 grants a glimpse on modern graphics cards architecture considering their programming and memory model. In Section 5, we describe the chosen implementation approach concerning algorithms and requirements when being applied on a graphics card. Section 6 reports our results before we conclude in Section 7.

2 Related Work

Parallelizing the lattice basis reduction is not a new subject to research, dedicated algorithms can be found in [34], [35], [11] and [16]. Recently, new attempts to parallelize lattice basis reduction algorithms were published which mainly focus on the SE-algorithm, respectively the BKZ-algorithm.

For the parallel variant of the SE-algorithm [4], POSIX threads are used to make an effective use of recent multicore computer architectures. In experiments with sparse and dense lattice bases, the algorithm shows a speed-up factor of about 1.75 for a 2-thread and close to factor 3 for the 4-thread version.

Many variants of shortest lattice vector enumeration, virtually the main part of the BKZ which targets the exact SVP, were ported to graphics cards: The lattice vector enumeration [13], the simple sample reduction [30], and an approach including extreme pruning [20]. But in order to obtain reasonable advances in performance each GPU implementation still requires a strong pre-reduction of the lattice basis and hence a fast LLL-algorithm that runs on a CPU to this day.

Further, another work also considered FPGAs for lattice enumeration [8].

3 Mathematical Background

In linear algebra, a lattice in $\mathbb{R}^n$ is a discrete, additive, Abelian subgroup of $\mathbb{R}^n$ consisting of points. The property being discrete implies that there are no cluster points but all points have a minimum *Euclidean* distance from each other.

Definition 1 (Lattice). *Let $b_1, b_2, \ldots, b_k \in \mathbb{R}^d, k \leq d$ linear independent, the set*

$$\mathcal{L} = \left\{ u \in \mathbb{R}^d | u = \sum_{i=1}^{k} a_i b_i, a_i \in \mathbb{Z} \right\}$$

is called a lattice.

Hence, every lattice $\mathcal{L}$ can be represented by a set $\mathcal{B} = \{b_1, b_2, \ldots, b_k\}$ of column vectors. We call $\mathcal{B}$ the basis of the lattice $\mathcal{L}$, thus $\mathcal{L}(\mathcal{B})$ is the set of all finite, integer linear combinations of the basis vectors b_i.

A basis $\mathcal{B}$ is never unique. Every basis $\mathcal{B}$ can be transformed into another basis $\mathcal{B}'$ such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}')$ while permuting columns, multiplying a column by -1 or adding an integer multiple of one vector to another vector. These operations are also referred to as a *unimodulary transformation*. More detailed information on lattices and their properties can be found in [9].

3.1 Problems in Lattices

The lattice theory is concerned with three major problems that are very briefly discussed in the following.

Shortest and Approximated γ-Shortest Vector Problem. Let $\mathcal{L}$ be a d-dimensional lattice and corresponding basis $\mathcal{B} \in \mathbb{R}^{d \times k}$. By the shortest vector problem (SVP), one obtains the basis $\mathcal{B}$ as input and searches for a vector with Euclidean length λ_1 (or $\gamma\lambda_1$ in case of an approximation). Here, λ_1 is the minimal radius of a ball enclosing the origin in $\mathcal{L}$ that contains the shortest lattice basis vector (first successive minimum).

Closest and Approximated γ-Closest Vector Problem. Let $\mathcal{L}$ be a d-dimensional lattice and corresponding basis $\mathcal{B} \in \mathbb{R}^{d \times k}$. By the closest vector problem (CVP) one searches a vector u that is the closest to a given target vector t, meaning $\|u - t\| = \min_{v \in \mathcal{L}} \|v - t\|$, respectively $\|u - t\| = \gamma \cdot \min_{v \in \mathcal{L}} \|v - t\|$ for the γ-approximated version.

Shortest and Approximated γ-Shortest Basis Problem. Let $\mathcal{L}$ be a d-dimensional lattice and corresponding basis $\mathcal{B} \in \mathbb{R}^{d \times k}$. By the shortest basis problem (SBP) one obtains the basis $\mathcal{B}$ as input and searches a basis such that $\|b_i\|_2 = \lambda_i, b_i \in \mathcal{B}$, respectively $\|b_i\|_2 = \gamma \cdot \lambda_i, b_i \in \mathcal{B}$ for the γ-approximated version.

3.2 Lattice Basis Reduction

The lattice basis reduction deals with the problem to find a short lattice basis for a given lattice basis (SBP). In practice, finding a shortest vector (SVP) in this

Algorithm 1. LLL-Algorithm

Input: Lattice basis $\mathcal{B} = (b_1, b_2, \ldots, b_k) \in \mathbb{R}^{d \times k}$ and reduction parameter δ with $\frac{1}{4} < \delta < 1$
Output: δ-LLL-reduced basis $\mathcal{B}$

```
Orthogonalization: Compute Gram-Schmidt coefficients μ_{i,j}
i = 2
while i ≤ k do
    Size Reduction: Size reduce the vector b_i
    if δ ‖b*_{i-1}‖_2^2 < ‖b*_i‖_2^2 + μ_{i,i-1}^2 ‖b*_{i-1}‖_2^2 then
        Basis Permutation: Swap basis vectors b_i and b_{i-1}
        Orthogonalization: Update Gram-Schmidt coefficients μ_{l,j} for l > i
        i = max(i − 1, 2)
    else
        i = i + 1
    end if
end while
```

basis is of particular importance. In 1982 Lenstra, Lenstra and Lovász proposed the first lattice basis reduction [21] that terminates in polynomial runtime, according to the lattice dimension, which is known as the *LLL-algorithm* named according to its inventors. The LLL-algorithm iteratively works in stages where each such stage processes the vector b_i. During the process, the algorithm can return to a certain stage several times before it terminates. A single stage consists of the following operations. First, the vectors b_i are reduced in their size by the preceding vectors $b_{i-1}, b_{i-2}, \ldots, b_1$ according to the Gram-Schmidt coefficients, which are defined by

$$\mu_{i,j} := \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}, \quad i > j$$

At beforehand, the orthogonal versions b_i^* of the basis vectors b_i are determined by

$$b_1^* := b_1; \quad b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*.$$

Afterwards it is checked if the swap condition, the so-called Lovász condition, is true involving the orthogonal version b_i^* of b_i. If so, then vector b_i is interchanged with vector b_{i-1}. Being in stage i the basis vectors $b_{i-1}, b_{i-2}, \ldots, b_1$ are already *LLL-reduced* by which means the vectors are

1. *Size reduced*: $|\mu_{i,j}| \leq \frac{1}{2}, \quad 1 \leq j < i \leq k$ and satisfy the
2. *Lovász condition*: $\delta \left\|b_i^*\right\|_2^2 \leq \left\|b_{i+1}^*\right\|_2^2 + \mu_{i+1,i}^2 \left\|b_i^*\right\|_2^2$ for $1 \leq i \leq k, \frac{1}{4} < \delta < 1$.

4 Computations on Graphics Cards

General-purpose computing on graphics processing units (GPGPU) is the shift of computations that are traditionally handled by the *central processing unit*

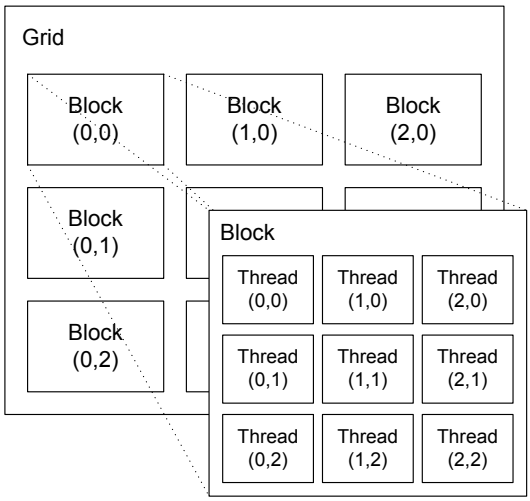


Fig. 1. CUDA thread hierarchy

(CPU) or *host* processor, to the *graphics processing unit* (GPU), also denoted as *device*. In this paper, we focus on nVidia GPUs and CUDA [25,26] that can be programmed with *C for CUDA*, a C language derivative with special extensions.

The main unit of the device is the multiprocessor which is a set of a number of stream processors or CUDA cores (the number depends on the generation and the model) that share memory, caches, and an instruction unit. The multiprocessor creates, manages, and executes *threads* in hardware. As Figure 1 shows, a thread in CUDA is the smallest unit of parallelism that is executed concurrently with other threads (*warps*) on the hardware. Threads are organized in a *thread block*, a group of threads in which the threads can communicate with each other and synchronize their state. A group of thread blocks is called a *thread grid*. A thread grid forms the execution unit in the CUDA model since it is not possible to execute a thread or thread block solely.

Threads in the CUDA programming model can access data from various memory spaces that differ in size and access time. The CUDA memory model (Fig. 2) describes the accessible memory spaces from the view point of the thread. At lowest level, a thread has read and write access to its own *registers* and additionally its own copy of *local memory*. Threads within the same block have read and write access to a *shared memory* on the next higher level. Beyond the block, all threads can have read and write access to the largest memory space, the *global memory*. Beside the global memory, there are two further spaces that are read-only, the *constant memory* and the *texture memory*. Usually, memory spaces that are shared by threads contain potential hazards of conflicts such as *read-after-write*, *write-after-read*, or *write-after-write*. Thus, the programming model implements a barrier by defining a synchronization instruction. As a consequence, a large number of divergent threads (*i.e.* threads which follow a different execution flow of an algorithm) require frequent synchronization, reducing the

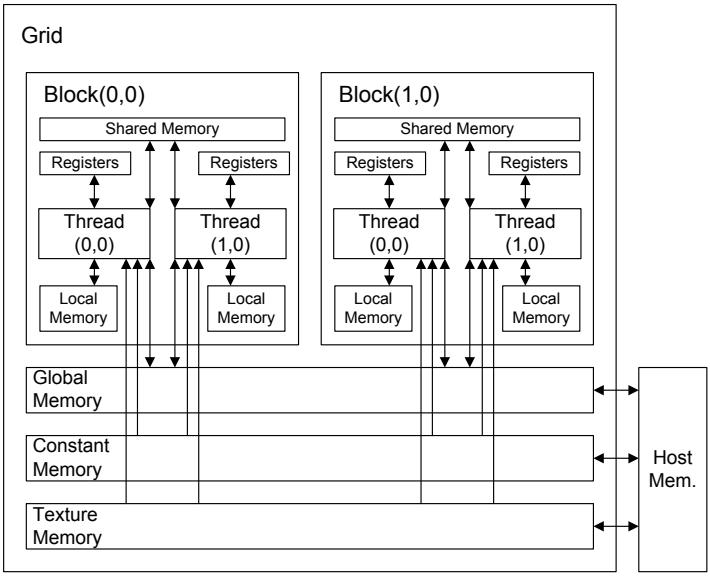


Fig. 2. CUDA memory model

overall computation time of the entire systems due to wait cycles. Accesses to the global memory are crucial for the overall performance due to its latency. But most of that latency can be hidden if there are enough independent arithmetic instructions that are executed while waiting for the access to complete.

5 Lattice Basis Reduction on Graphics Cards

According to Algorithm 1, lattice basis reduction has three main phases: first the *basis orthogonalization* (computation of Gram-Schmidt coefficients), second the *size reduction* of the basis and third the *basis permutation*.

5.1 Parallel Orthogonalization

For parallel orthogonalization the *QR decomposition* is a tool that generates an orthogonal basis in the sense of an unimodulary transformation. As this name indicates, the QR decomposition disjoints an intended matrix $A \in \mathbb{R}^{d \times k}$ into two matrices Q and R. Here, Q is not of further interest but R, an upper triangular matrix, which contains the Gram-Schmidt coefficients. There are several methods to compute the QR decomposition. In this paper, we focus on a parallel variant of the Givens rotations. Detailed informations on the QR decomposition and the corresponding methods can be found in [9].

Givens Rotations are, beside Householder reflections and many other, an alternative way to compute the QR decomposition. Whereas Householder reflections insert zeros column-wise by updating the entire basis matrix based upon

a matrix-matrix multiplication, Givens rotations convert step by step the basis matrix $\mathcal{B}$ into the upper triangular form by inserting lower triangular zeros from the most left column to the most right column. One Givens rotation is represented by an usual identity matrix of the form

$$G(i,j)^T\mathcal{B} = \begin{pmatrix} 1 & \dots & 0 & & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & & \vdots & & \vdots \\ 0 & \dots & c & & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & & \vdots & & \vdots \\ 0 & \dots & -s & & \dots & c & \dots & 0 \\ \vdots & & \vdots & & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & & \dots & 0 & \dots & 1 \end{pmatrix}^T \mathcal{B},$$

where $g_{i,i} = g_{j,j} = c = \frac{b_{i,i}}{r}$, $g_{i,j} = s = \frac{b_{j,i}}{r}$, $g_{j,i} = -s$, and $r = \sqrt{b_{i,i}^2 + b_{j,i}^2}$, that is multiplied from the left to the basis matrix $\mathcal{B}$.

The Givens rotations matrix implies that only two rows of $\mathcal{B}$ are affected at once, namely the i_{th} and j_{th} row. The order is important to obtain the correct result, columns have to be zeroized step by step from left to right. Thus, one has to follow a sequential order which can be partially parallelized. Nevertheless, a parallelized variant exploits the fact that rows that are already processed by their preceding rows can also be taken into account. Algorithm 2 shows such a parallelized version to compute Givens rotations. Our implementation of this algorithm considered two levels of optimizations:

One-Block-Per-Associated Rows (OBPA). This approach realizes an effective way to implement the Givens rotations whereas one thread block is responsible for two affected rows implied by a Givens rotation to insert a single zero. First the kernel computes the values c, s and then updates the remaining entries of each row.

OBPA with Combined Stages (OPBA n-staged). This optimizations exploits the fact that consecutive iterations of the outer loop largely involve the same rows (cf. Alg. 2). Accesses to the global memory of the graphics card are a crucial factor concerning the overall performance. Hence, one can reduce the access time to the global memory by almost n, if n iterations (stages) are combined, while the upper row values reside in the threads' registers as long as they are processed together with following lower row values. For instance, the first row is processed together with the second in the first stage and with the third row in the consecutive second stage, and thus the updated values of the first row are stored back first after the second processing with row three, that is, the first two iterations of the outer loop are processed together (combined stages).

As a last step, the obtained matrix R needs to be transformed into Gram-Schmidt coefficients that are required by the lattice reduction. This transformation is simply given by a CUDA kernel that divides each element $r_{i,j}$ of each row i by its respective diagonal element $r_{i,i}$.

Algorithm 2. Parallel Givens Rotations

Input: Lattice basis $\mathcal{B} = (b_1, b_2, \ldots, b_k) \in \mathbb{R}^{d \times k}$
Output: $R = [r_{i,j}]_{1 \leq i,j \leq k}$

1: $R = \mathcal{B}$
2: **if** $d = k$ **then**
3: $\quad l = k - 1$
4: **else**
5: $\quad l = k$
6: **end if**
7: **for** $e = 3$ to $d + l$ **do**
8: $\quad$ **for** $i = \max(1, e - d)$ to $\frac{e-1}{2} - \max(0, \frac{e-1}{2} - k)$ **parallel do**
9: $\quad\quad j = e - i$
10: $\quad\quad$ Compute c, s: $r' = \sqrt{r_{i,i}^2 + r_{j,i}^2}$, $c = \frac{r_{i,i}}{r'}$ and $s = \frac{r_{j,i}}{r'}$
11: $\quad\quad R = G(i, j)^T R$
12: $\quad$ **end for parallel**
13: **end for**

5.2 Parallel Basis Size Reduction

A straightforward parallel method in order to reduce a basis in size (that can easily adapted for use with graphics cards) does not exist. This is due to the circumstance that a basis vector b_i has to be reduced by a previous vector b_j after reducing b_j itself. Commonly, previous works [35] use the pattern which was introduced for the Givens rotations (lines $7-9$, Alg. 2) but starting at basis vector b_k. We propose a novel pattern involving Algorithm 3 that optimally fits the architecture of graphics cards. Beforehand, the matrix R is transposed to facilitate a fast global memory access. However, we decoupled the size reduction of the basis from that of the Gram-Schmidt coefficients. Hence, we first reduce the Gram-Schmidt coefficients, except the coefficient that is currently used for the size reduction (line 5, Alg. 3 is replaced by $\mu_{i,j} = \lceil \mu_{i,j} \rfloor$), and compute

Algorithm 3. Parallel Basis Size Reduction

Input: Lattice basis $\mathcal{B} = (b_1, b_2, \ldots, b_k) \in \mathbb{R}^{d \times k}$ and Gram-Schmidt coefficients $[\mu_{i,j}]_{1 \leq i,j \leq k}$
Output: Size reduced basis $\mathcal{B}$

1: **for** $e = 1$ to $k - 1$ **do**
2: $\quad$ **for** $i = k$ to $k - e + 1$ **parallel do**
3: $\quad\quad j = i - 1$
4: $\quad\quad b_i = b_i - \lceil \mu_{i,j} \rfloor \, b_j$
5: $\quad\quad \mu_{i,j} = \mu_{i,j} - \lceil \mu_{i,j} \rfloor$
6: $\quad\quad$ **for** $l = 1$ to $j - 1$ **do**
7: $\quad\quad\quad \mu_{i,l} = \mu_{i,l} - \lceil \mu_{i,j} \rfloor \, \mu_{i,j}$
8: $\quad\quad$ **end for**
9: $\quad$ **end for parallel**
10: **end for**

the nearest integer ($\lceil \bullet \rfloor$) at a time. Roughly speaking, we start by reducing the Gram-Schmidt coefficients vector of the most right basis vector with help of the left Gram-Schmidt vectors.

Next, we reduce the basis with help of the pre-computed Gram-Schmidt coefficients in $k-1$ iterations. Therefore, we calculate a weighted sum row-wise, involving the respective basis vector entries and their dedicated integer versions of the Gram-Schmidt coefficients and subtract it from the corresponding basis vector entry.

5.3 Parallel Basis Permutation

The so-called *All-Swap* [33] lattice basis reduction intends to process as much as possible of the entire basis with respect to orthogonalization, size reduction and permutation by swapping. The basis is kept in both representations (precise and floating point arithmetic) to eliminate the need for type conversions. The algorithm works iteratively in competitive alternating phases, an *odd* and an *even* phase. These phases facilitate performing as much vector swaps as possible on disjoint vector sets. To cushion the appearance of round-off errors, the approximated basis is recomputed from the exact basis each time and thus concurrently the orthogonalization and the size reduction is performed each time on the entire lattice basis. The original All-Swap floating point algorithm was proposed by Heckler and Thiele [12] (referred to as *original All-swap* in the following).

Here, we introduce a variant from that we expect a better reduction quality due to a higher number of swap operations, the *sorted All-swap phase* which is a generalized variant of the All-swap phase. Instead of swapping two adjoining vectors by which means sorting them according to their squared 2-norms $\|b_i^*\|_2^2$, blocks of size 2^l, with $2^l \leq k$, vectors will be sorted. Obviously, $l = 1$ will give the original All-swap phase. Our sorted approach is represented by Algorithm 4. The reduction parameter δ' is deduced from the original δ that is included in the LLL-algorithm. Therefore, it is possible to state a simplified Lovász condition such that

$$\delta \left\|b_i^*\right\|_2^2 \leq \left\|b_{i+1}^*\right\|_2^2 + \mu_{i+1,i}^2 \left\|b_i^*\right\|_2^2 \Rightarrow \left\|b_i^*\right\|_2^2 \leq \delta' \left\|b_{i+1}^*\right\|_2^2,$$

with $|\mu_{i,j}| \leq \frac{1}{2}$ and $\delta' = (\delta - \frac{1}{4})^{-1}$.

In our implementation we applied a partitioned insertion sort algorithm according to the number of threads. The correctness of the presented algorithm can be obtained by observing the lattice determinant (*i.e.* the matrix determinant in the case of square matrices) which remains invariant during the entire reduction process. Note that a parallel lattice basis reduction algorithm depends on the reduction parameter (δ') contrary to a sequential algorithm. However, for low δ' values dependent on the given lattice, the process can potentially lead to an endless loop due to precision issues. For higher δ' values, the algorithm terminates with high probability.

Algorithm 4. Sorted All-Swap Lattice Basis Reduction

Input: Lattice basis $\mathcal{B} = (b_1, b_2, \ldots, b_k) \in \mathbb{R}^{d \times k}$, reduction parameter δ' with $\delta' > \frac{4}{3}$ and block-size parameter l
Output: δ'-All-swap-reduced basis $\mathcal{B}$

1: $phase = 0$
2: **while** sorting is possible for any block in phase **do**
3: Approximate basis $\mathcal{B}' = (\mathcal{B})'$
4: Compute Gram-Schmidt coefficients $\mu_{i,j}$
5: Size reduce the basis $\mathcal{B}$
6: Split the basis into m blocks of size 2^l starting with $b_{1+phase \cdot 2^{l-1}}$
7: **for** $i = 1$ to m **parallel do**
8: Using an appropriate sorting algorithm to sort the block $b_{2^l(i-1)+phase \cdot 2^{l-1}+1}, \ldots, b_{2^l i+phase \cdot 2^{l-1}}$ by its squared 2-norms, *i.e.* $\|b_r^*\|_2^2 \leq \delta' \|b_s^*\|_2^2, \quad 2^l(i-1) + phase \cdot 2^{l-1} + 1 \leq r < s \leq 2^l i + phase \cdot 2^{l-1}$
9: $phase = phase \oplus 1$
10: **end for parallel**
11: **end while**
12: Approximate basis $\mathcal{B}' = (\mathcal{B})'$
13: Compute Gram-Schmidt coefficients $\mu_{i,j}$
14: Size reduce the basis $\mathcal{B}$

6 Results

For our experiments, we used an nVidia GTX 280 graphics cards with 1 *GiB* video RAM and an Intel Core 2 Quad Q9550 at 3.6 *GHz* running Windows 7 64-bit. The results were obtained using the CUDA toolkit and SDK *3.2* and the CUDA driver *260.89*.

Experiments have shown that the OPBA n-staged method for Givens rotations achieves a significant speed-up for $n = 2$, respectively $n = 4$ combined stages. Combining more stages results in a negative effect concerning runtime due to the severely increased sequential effort per thread. The OPBA methods is executed complying with Algorithm 2, meaning several sufficient kernel calls dependent on the magnitude of the basis matrix, the number of blocks according to the number of affected rows. It turned out that 64 threads per block are optimal.

To provide reasonable results for the full lattice basis reduction, we consider lattice bases obtained from the *TU Darmstadt Lattice Challenge*[1], random lattices (entries are chosen randomly within a certain range), and random lattices in Hermite normal form. This choice also satisfies the requirement that graphics cards only support double floating point precision yet.

For the *CLB-200* (a lattice basis challenge of dimension 200), we obtain a similar reduction quality for All-swap algorithms using different δ' values as shown in Table 1. Generally, we observe that methods that perform more swaps than the original All-swap require much less runtime and deliver a similar

[1] Available on `http://www.latticechallenge.org`

Table 1. Results for different All-swap implementations processing the CLB-200. Here, v_s is the shortest vector within the lattice basis with $\|v_s\|_2 < n = 30$ to win CLB-200. Blocked-sorted $2^l = 2$ corresponds to the original All-swap method.

l	δ'	Determinant	Overall size red.	$\|v_s\|_2$	Runtime
1	1.34	$2.0589 \cdot 10^{44}$	280	19.03	3.37 s
2	1.6	$2.0589 \cdot 10^{44}$	178	18.84	2.11 s
3	3.0	$2.0589 \cdot 10^{44}$	75	21.79	0.89 s
4	2.55	$2.0589 \cdot 10^{44}$	–	9.22	63.9 s

reduction quality at a time. Among all of our results, we found a shortest vector with the Euclidean length 9.22 by application of the block-sorted All-swap with block-size 16, $\delta' = 2.55$ after a computation time of 63.9 seconds. The CLB-500 challenge can be performed with a similar setup, more precisely block-size 16, $\delta' = 3.05$ and a computation time of 104.63 seconds with Euclidean norm of 62.94 ($n = 63$). Although this is not an optimal value since it is very close to n, it still provides a significant reduction with respect to runtime when employing our lattice basis reduction. Compared to runtime results from NTL[2], as shown in Figure 3, that involves the Schnorr-Euchner algorithm in double floating point precision using Givens rotations (`G_LLL_FP()` which is the sequentially executed counterpart on a CPU), our implementation achieves a speed-up of about 12.5 ($l = 1$) and 18.6 ($l = 2$) on average considering the challenge lattices. In general, we observe an increasing speed-up factor according to higher lattice basis dimensions. Moreover, the reduction quality is nearly about the same as depicted in Figure 4(a) which compares the lengths of the shortest vectors found by each algorithm. Note that this metric was chosen for the sake of simplicity, but it also represents the different reduced bases as a whole. Second, note that other δ' values lead to other vector length with almost the same runtime, hence the NTL does not provide better results in general given a similar δ'. Figure 4(b) shows the runtime performance for both randomly chosen lattice bases and randomly chosen lattice bases in Hermite normal form using a logarithmic scale.

As can be seen, the reduction of these types of lattices on the GPU even outperform the reduction of the challenge lattice bases. The NTL copes better with random lattices than with random lattices in Hermite normal form. To provide a fair comparison between CPU-based and GPU-based lattice basis reduction, we put the costs of two computing systems each for CPU and GPU-based lattice basis reduction into relation. In this model, we assume €500 for a system equipped with a powerful CPU, such as an Intel quad core, and onboard-graphics and €680 for a system containing a recent nVidia GTX graphics card but populated with a low-end desktop processor. Normalizing the speed-up of the GPU-based implementation according to the higher cost of its computing system, we still have a 13.7 times higher performance compared to the corresponding CPU-based system for the same amount of financial investment.

[2] All measurements were performed on the same system as presented above.

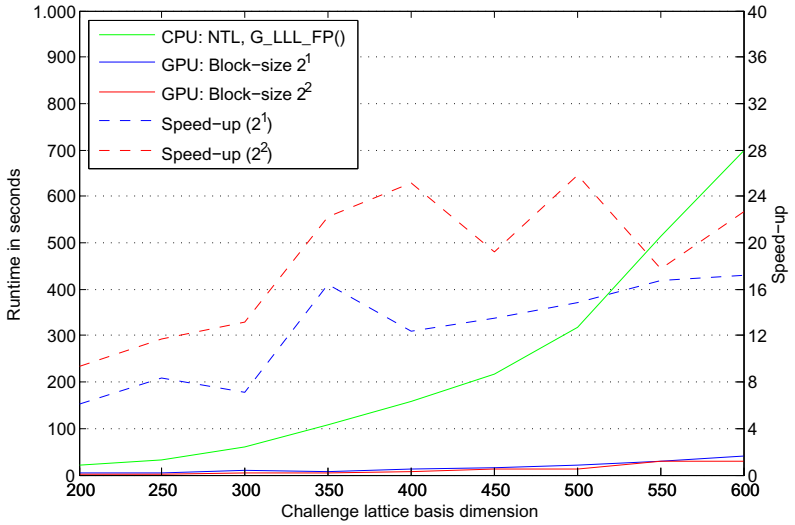
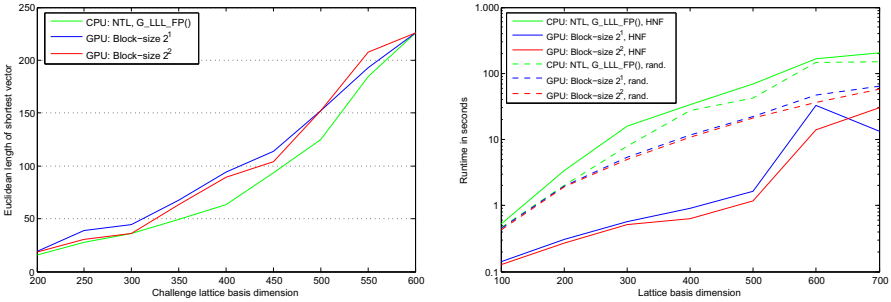


Fig. 3. Runtime for challenge lattice bases with $\delta' = 1.34$ ($l = 1$), $\delta' = 1.55$ ($l = 2$) and $\delta = 0.99$ (NTL)



(a) Euclidean length of the shortest vectors found

(b) Runtime for random lattice bases and random lattice in Hermite normal form

Fig. 4. Runtime measurements were done using $\delta' = 1.34$ ($l = 1$), $\delta' = 1.55$ ($l = 2$) and $\delta = 0.99$ (NTL)

7 Conclusion and Future Work

In this paper we presented the first implementation of a full lattice basis reduction on graphics cards. Due to the memory model that enables the many-core based GPU to compute the established algorithms massively in parallel, we achieved promising results with respect to other CPU-based implementations, such as NTL. We introduced a variant of the All-swap algorithm that delivers better reduction results with decreased runtime with respect to challenge lattice bases. Our implementation can also be used to find short vectors, however at

the cost of a higher runtime. Still, our implementation requires some trials due to find an optimal reduction parameter δ'.

Future work involves the OpenCL framework that offers quadruple floating point precision in the next versions. Thus, it would become possible to reduce either lattice bases with very high dimensions or lattice bases consisting of large entries which is, as of now, restricted by the current GPU generation. Alternatively, future work could apply the proposed approach as a pre-reduction for lattice enumeration.

References

1. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: STOC 1996: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 99–108. ACM, New York (1996)
2. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: STOC 1997: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 284–293. ACM, New York (1997)
3. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009)
4. Backes, W., Wetzel, S.: A Parallel LLL using POSIX Threads. Tech. rep., Dept. of Computer Science, Stevens Institute of Technology (2009), dIMACS Technical Report 2008-12
5. Brakerski, Z., Goldwasser, S., Kalai, Y.: Circular-Secure Encryption Beyond Affine Functions. Cryptology ePrint Archive, Report 2009/485 (2009), http://eprint.iacr.org/
6. Buchmann, J., Lindner, R., Rückert, M., Schneider, M.: Explicit hard instances of the shortest vector problem. Cryptology ePrint Archive, Report 2008/333 (2008), http://eprint.iacr.org/
7. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient Public Key Encryption Based on Ideal Lattices. Cryptology ePrint Archive, Report 2009/285 (2009), http://eprint.iacr.org/
8. Detrey, J., Hanrot, G., Pujol, X., Stehlé, D.: Accelerating Lattice Reduction with FPGAs. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 124–143. Springer, Heidelberg (2010)
9. Gentle, J., Härdle, W., Mori, Y.: Handbook of Computational Statistics. Springer, Heidelberg (2004)
10. Goldreich, O., Goldwasser, S., Halevi, S.: Collision-Free Hashing from Lattice Problems (1996)
11. Heckler, C.: Automatische Parallelisierung und parallele Gitterbasisreduktion. Ph.D. thesis, Universität des Saarlandes, Saarbrücken (1995)
12. Heckler, C., Thiele, L.: Parallel Complexitiy of Lattice Basis Reduction and a Floating-Point Parallel Algorithm. In: Reeve, M., Bode, A., Wolf, G. (eds.) PARLE 1993. LNCS, vol. 694, pp. 744–747. Springer, Heidelberg (1993)
13. Hermans, J., Schneider, M., Buchmann, J., Vercauteren, F., Preneel, B.: Parallel Shortest Lattice Vector Enumeration on Graphics Cards. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 52–68. Springer, Heidelberg (2010)
14. Hinek, M.: Lattice Attacks in Cryptography: A Partial Overview. Tech. rep., School of Computer Science, University of Waterloo (2004)

15. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A Ring-Based Public Key Cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998)
16. Joux, A.: A Fast Parallel Lattice Basis Reduction Algorithm. In: Proceedings of the Second Gauss Symposium, pp. 1–15 (1993)
17. Joux, A., Stern, J.: Lattice Reduction: a Toolbox for the Cryptanalyst. Journal of Cryptology 11, 161–185 (1994)
18. Kerr, A., Campbell, D., Richards, M.: QR Decomposition on GPUs. Tech. rep., Georgia Institue of Technlogogy, Georgia Tech Research Institute (2009)
19. Khronos Group: The OpenCL Specification Version 1.1 (2011), http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf
20. Kuo, P.-C., Schneider, M., Dagdelen, O., Reichelt, J., Buchmann, J., Cheng, C.-M., Yang, B.-Y.: Extreme Enumeration on GPU and in Clouds. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 176–191. Springer, Heidelberg (2011)
21. Lenstra, A., Lenstra, H., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 261(4), 515–534 (1982)
22. 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)
23. Merkle, R., Hellman, M.: Hiding information and signatures in trapdoor knapsacks. IEEE Transactions on Information Theory 24, 525–530 (1978)
24. Nguyen, P.Q., Stehlé, D.: An LLL Algorithm with Quadratic Complexity. SIAM Journal on Computing 39(3), 874–903 (2009)
25. nVidia: NVIDIA CUDA Development Tools (2010), http://developer.download.nvidia.com/compute/cuda/3_2/docs/Getting_Started_Windows.pdf
26. nVidia: NVIDIA CUDA Programming Guide (2010), http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Programming_Guide.pdf
27. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: STOC 2009: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 333–342. ACM, New York (2009)
28. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM 56(6), 1–40 (2009)
29. Rivest, R., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21, 120–126 (1978)
30. Schneider, M., Göttert, N.: Random Sampling for Short Lattice Vectors on Graphics Cards. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 160–175. Springer, Heidelberg (2011)
31. Schnorr, C., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Program. 66(2), 181–199 (1994)
32. Shoup, V.: NTL: A Library for doing Number Theory, http://www.shoup.net/ntl/
33. Villard, G.: Parallel lattice basis reduction. In: ISSAC 1992: Papers from the International Symposium on Symbolic and Algebraic Computation, pp. 269–277. ACM, New York (1992)
34. Wetzel, S.: An Efficient Parallel Block-Reduction Algorithm. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 323–337. Springer, Heidelberg (1998)
35. Wiese, K.: Parallelisierung von LLL-Algorithmen zur Gitterbasisreduktionen. Master's thesis, Universität des Saarlandes (1994)

Breaking DVB-CSA

Erik Tews[1], Julian Wälde[1], and Michael Weiner[2]

[1] Technische Universität Darmstadt, Fachbereich Informatik,
Hochschulstraße 10, 64289 Darmstadt
{e_tews,jwaelde}@cdc.informatik.tu-darmstadt.de
[2] Technische Universität München
michaelweiner@mytum.de

Abstract. Digital Video Broadcasting (DVB) is a set of standards for digital television. DVB supports the encryption of a transmission using the Common Scrambling Algorithm (DVB-CSA). This is commonly used for PayTV or for other conditional access scenarios. While DVB-CSA support 64 bit keys, many stations use only 48 bits of entropy for the key and 16 bits are used as a checksum. In this paper, we outline a time-memory-tradeoff attack against DVB-CSA, using 48 bit keys. The attack can be used to decrypt major parts a DVB-CSA encrypted transmission online with a few seconds delay at very moderate costs. We first propose a method to identify plaintexts in an encrypted transmission and then use a precomputed rainbow table to recover the corresponding keys. The attack can be executed on a standard PC, and the precomputations can be accelerated using GPUs. We also propose countermeasures that prevent the attack and can be deployed without having to alter the receiver hardware.

1 Introduction

Digital Video Broadcasting (DVB) is a set of standards for digital television in Europe. It has been standardized by the European Telecommunication Standardization Institute (ETSI) in 1994. DVB defines multiple standards in the field of digital television; well-known standards include terrestrial (DVB-T) [5], satellite (DVB-S) [3], and cable television (DVB-C) [4]. However, there are many more standards, e.g. for data broadcasting.

DVB supports encryption of payload using the proprietary Common Scrambling Algorithm (DVB-CSA). The main use cases are PayTV and conditional access for TV stations that only have regionally limited broadcasting rights. Other use cases are possible as well, e.g. encryption of IP-over-satellite data traffic [7].

DVB-CSA was not intended for public disclosure and manufacturers implementing it need to sign a non-disclosure agreement to get access to the specifications [1]. Only a few details about the structure of the encryption scheme were known from the standard [2], a publication [8], and a patent application [6]. The breakthrough leading to the full disclosure of DVB-CSA took place when FreeDec, a software implementation of DVB-CSA, appeared on the Internet in

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 45–61, 2012.

2002. This implementation was reverse-engineered to extract the missing details of the cipher, such as the S-Box used.

The first academic publication analyzing DVB-CSA appeared one year later [13]. Other publications we found about DVB-CSA consider physical attacks, i.e. fault attacks [14] and side-channel attacks [9] or only analyze the stream cipher part of DVB-CSA [12], while DVB-CSA also contains a block cipher (see Section 2). However, those attacks do not work in a real-life scenario.

Common PayTV setups consist of four core components: a Smart Card, a Conditional Access Module (CAM), a set-top box and the television. The Smart Card is personalized to the PayTV subscriber and provides the DVB-CSA keys, which are changed frequently. It is able to compute the DVB-CSA keys based on a secret stored on the card and control messages from the TV station. The Conditional Access Module is the interface between the Smart Card and the set-top box. The CAM is either a PCMCIA card connected to the set-top box over the Common Interface (CI) or it is integrated into the set-top box. The set-top box decodes the MPEG stream and forwards it to the television, and the television finally displays the video.

All public practical attacks on encrypted DVB streams we know consider attacking the DVB-CSA key derivation scheme – this includes physical attacks against SmartCards as well as Card Sharing, i.e. distributing the DVB-CSA keys generated by a SmartCard to multiple users.

1.1 Our Contribution

To the best of our knowledge, we provide the first practical attack on DVB-CSA itself. It can be used to determine DVB-CSA session keys within a few seconds, regardless of the key derivation scheme. We use pre-computed rainbow tables [10] for our attack and reduce the key space by exploiting the fact that most 64-bit DVB-CSA keys use 16 of the key bits as a checksum.

This paper is organized as follows: We first introduce the reader to the DVB-CSA encryption algorithm in Section 2. In Section 3, we outline that DVB-CSA keys (64 bit) usually contain only 48 bits of entropy and 16 bit of the key are used as a checksum. In Section 4, we examine the MPEG2 Video broadcasted by many TV stations, and show that it usually contains constant plaintexts. In Section 5, we use this fact combined with the reduced key space to show that a time-memory tradeoff can be used against DVB-CSA, using *rainbow tables*. In Section 6, we show how tools that generate such tables can be efficiently implemented on CPUs and GPUs, and benchmark their performance. In Section 7, we suggest appropriate parameters for the generation of these tables. In Section 8, we present experimental results with a small table. In Section 9, we outline the overall attack scenario. In Section 10 we suggest countermeasures that prevent the attack and can be implemented with very low costs. We finally conclude in Section 11.

2 DVB-CSA in a Nutshell

DVB-CSA is the symmetric cipher used to protect content of MPEG2 Transport Streams in DVB. To transmit multiple audio, video or general data streams on a single transponder, MPEG Transport Stream (MPEG TS) encapsulates all data streams in *cells* of 188 bytes. These cells consist of a 4 byte header and 184 bytes of payload. Optionally, an extended header can be embedded, reducing the payload size by the size of the extended header.

A flag in the MPEG TS header indicates whether the packet is unencrypted, or encrypted with the *even* or *odd* key. Usually, only one key is used, while the other key is updated by the CAM/SmartCard. For encrypted cells, only the payload is encrypted. The header or optional extended header is never encrypted.

DVB-CSA works in 2 passes. The first pass splits the plaintext of the payload into blocks of 64 bit length and a remainder that is smaller than 64 bit; all blocks except this remainder are then encrypted with a custom block cipher in CBC mode, using reverse block order and all zero initialization vector. In the second pass, a stream cipher using the first block (last block in the order used with the block cipher) as initialization vector encrypts all data again, except the first block. Note that DVB-CSA does not randomize the ciphertexts: Equal plaintexts are always mapped to the same ciphertexts.

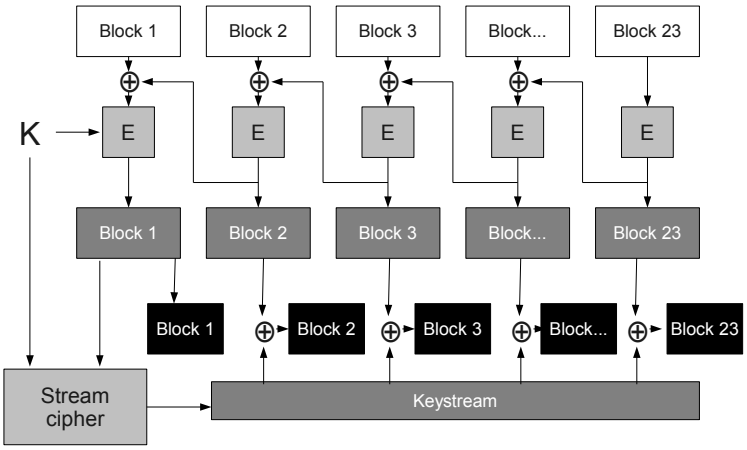


Fig. 1. DVB-CSA structure

2.1 The DVB-CSA Block Cipher

We will concentrate on the custom block cipher used by DVB-CSA as our attack focuses on the first block of the ciphertext which does not depend on the stream cipher (see Figure 1). We define the variables used for the block cipher as follows:

Definitions

KEY	the 64 bit KEY
K_i	the ith byte of the 64 bit key
KS[i]	the ith round key (1 byte)
$R[i]$	the ith byte of the 64 bit plaintext
S	a 8 bit permutation
P	a 8 bit permutation that only swaps bits

Key Schedule. The block cipher encrypts a single block in 56 rounds without any kind of additional input or output transformation. The key schedule of the block cipher expands the 64 bit key into 56 8 bit round keys using the bit permutation from Table 1.

$X_6 = \text{KEY}$
for $i = 6 \rightarrow 0$ **do**
  $X_{i-1} \leftarrow \text{permute\_block}(X_i)$
end for
for $i = 0 \rightarrow 6$ **do**
  $KS[8 * i \ldots 8 * (i + 1) - 1] \leftarrow X_i \oplus (0x0101010101010101 * i)$
end for

Round Function. One round of the DVB-CSA block cipher consists of six 8 bit XORs, one S-Box lookup S (see the sourcecode in Appendix A) and a 8 bit bit-permutation P.

P permutes bit indices as in this table:

in	0	1	2	3	4	5	6	7
out	1	7	5	4	2	6	0	3

3 Usage of DVB-CSA

In spite of the fact that DVB-CSA works with 64 bit keys, we observed that only 48 bit of entropy are used for many TV stations. The fourth and the eigth byte of the key in this case are the sum of the previous three bytes modulus 256:

Table 1. permute_block(X)

swaps the bits in X as in this table (decimal notation):

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
00	19	27	55	46	01	15	36	22	56	61
10	39	21	54	58	50	28	07	29	51	06
20	33	35	20	16	47	30	32	63	10	11
30	04	38	62	26	40	18	12	52	37	53
40	23	59	41	17	31	00	25	43	44	14
50	02	13	45	48	03	60	49	08	34	05
60	09	42	57	24						

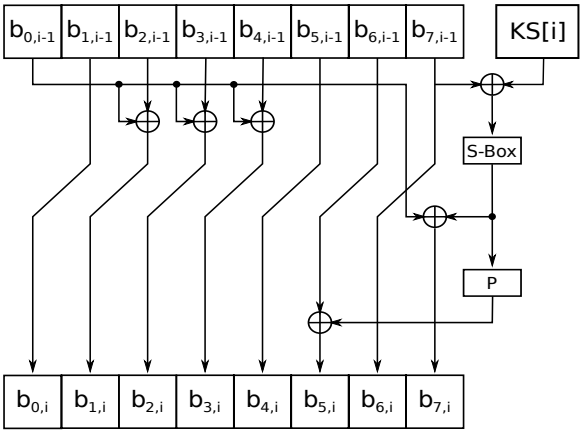


Fig. 2. DVB-CSA block cipher round function

k_0	k_1	k_2	$k_0 + k_1 + k_2 \bmod 256$	k_4	k_5	k_6	$k_4 + k_5 + k_6 \bmod 256$

This reduces the effort needed for an exhaustive search from 2^{64} to 2^{48} trial decryptions. This fact was not mentioned in previous academic publications [13,14,12] but is actually documented on the english Wikipedia (as of 2006)[1]. Because DVB-CSA is a non public standard that has been reverse engineered, we do not know whether these checksums are part of the specification or originate from cryptography export restriction.

Since 2^{48} trial decryptions are clearly possible for small corporations and even individuals, DVB-CSA poses more likely a hindrance than a perfect protection of the payload. All TV stations (broadcasted on Astra 19.2) we monitored change the DVB-CSA key every 7 to 10 seconds using a smart card based key distribution system instead of using one (then manually entered) key over a longer period of time. Some TV stations use a constant DVB-CSA key for a longer period, that is manually set at the receiver. This mode is called Basic Interoperable Scrambling System (BISS).

4 Recovering Plaintexts

In order to attack DVB-CSA, we began searching for a constant known plaintext in the MPEG-2 video data (H262). Because every bit of the ciphertext depends on every bit of the plaintext, we were looking for a repeating plaintext spawning a full MPEG-2 Transport Stream (TS) cell. Surprisingly, we found out that

[1] http://en.wikipedia.org/w/index.php?title=Common_Scrambling_Algorithm&diff=41583343&oldid=22087243

the video stream of many TV stations contains a lot of cells with a payload completely filled with zero-bytes. We had not expected this as MPEG-2 video uses various compression techniques to reduce the bandwidth, and a video stream which contains a lot of cells filled with zero bytes can be easily compressed.

In MPEG-2 video (H262, ISO 13818-2), the video compression codec used for many DVB variants supports so-called *stuffing bytes* – they are used to ensure a minimum bit rate. The ISO 13818-2 standard allows only zero bytes to be inserted between elements of the bitstream [11]. Since DVB-CSA is a completely deterministic encryption scheme: Encryptions of the same plaintext with the same key always result in the same ciphertext. Therefore, if at least two zero-filled frames are broadcasted during the lifetime of one key, these frames result in colliding ciphertexts. Cells completely filled with zero-bytes were the only constant plaintexts that are broadcasted very frequently. We can detect the corresponding encrypted cells by looking for repeating cells (collisions) in the encrypted video stream. We decided to assume that the most frequently colliding cell during a key lifetime corresponds to an encrypted cell filled with zeros.

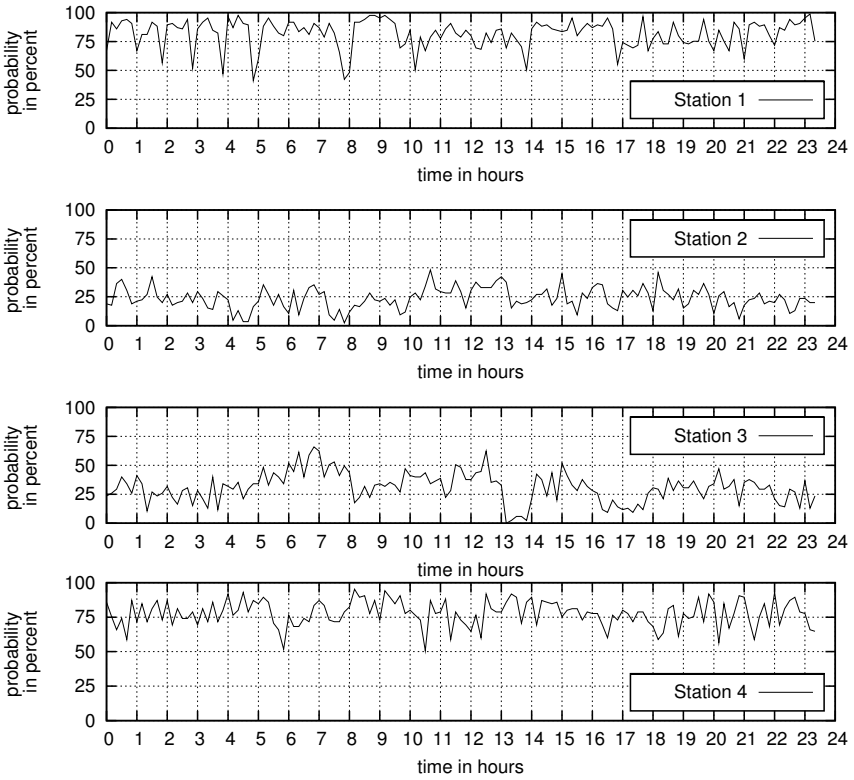


Fig. 3. Plaintext recovery success probability

Unfortunately, the occurrence of zero-filled cells heavily depends on the video content. For example, an interview with a person sitting in a chair without any movement in the background produces many zero-filled stuffing frames. In contrast, when old analog film that contains a lot of scratches and other artifacts is digitized, nearly no cells are filled with zero. We also checked that these cells usually occur when the data rate of the video stream slightly drops.

To make a quantitative statement about the occurrence of zero-filled cells, we measured the relative occurrence of collisions of zero-filled cells on unencrypted stations within intervals of 7 seconds. The detection of a zero-filled cell is considered successful if at least one collision is detected within an interval of seven seconds, and if the most frequently occurring collision actually corresponds to an all-zero cell. The evaluation is done with a granularity of 10 minutes on four popular and unencrypted TV stations airing on the *Astra 19.2* satellite for 24 hours.

This approach of analyzing unencrypted stations is equivalent to measuring how often ciphertext collisions can be found on an encrypted stream for which the key changes every 7 seconds, and evaluating how often these collisions actually decrypt to all-zero cells.

The results of this experiment can be found in Figure 3. We also found a station on *Astra 19.2* sending so many zero-filled cells that the recovery rate was 100% in our analysis interval.

This rate of intervals in which a zero cell appears most often is an *upper bound* to the success rate of any kind of attack, that is based on this plaintext recovery heuristic.

5 A Time Memory Tradeoff

Since DVB-CSA keys are changed frequently in most use cases, an attacker is interested in recovering DVB-CSA keys very fast. To break DVB-CSA, we use a time memory trade-off as described by Oechslin [10] to recover keys within seconds from a single known plain-text/cipher-text pair. Oechslin invented a general method known as *Rainbow Tables* to invert one way functions faster than by exhaustive search, but precomputations need to be made.

As a one-way function upon which a rainbow table can be built, we propose a mapping f that takes an 48 bit key (without the two checksum bytes) as input and returns the first 6 ciphertext bytes of an all zero cell encrypted with this key. Note that in order to compute f, only 23 calls of the block cipher are required (as described in Section 2, one cell consists of 23 blocks of 8 bytes and all of them need to be processed). As reduction function R_i one could simply XOR the input of f with i.

If we can find the first 6 bytes of a ciphertext c that corresponds to a zero-filled plaintext cell, then $f^{-1}(c)$ is a key (without checksum bytes) that encrypts a zero-filled cell to that specific ciphertext. With some luck it is also the current decryption key (multiple keys could exist encrypting a zero-filled cell to these first 6 bytes of the specific ciphertext).

5.1 Construction

To generate such a rainbow table, we need to generate many chains of length t of the form:

$$k_0 \overset{R_0 \circ f}{\longrightarrow} k_1 \overset{R_1 \circ f}{\longrightarrow} k_2 \cdots k_{t-3} \overset{R_{t-3} \circ f}{\longrightarrow} k_{t-2} \overset{R_{t-2} \circ f}{\longrightarrow} k_{t-1}$$

where $f : k \rightarrow csablock_k^{23}(0)$. R_i reduces the 64 bit output of *csablock* to a new 48 bit input for f and also is different for each i:

$$R_i(x) := x \oplus (i || table_{id})$$

where $table_{id}$ is an 16 bit unsigned and i is an 32 bit unsigned integer in little-endian[2] representation and $||$ denotes concatenation.

For these chains we only store the head k_0 and the tail k_{t-1} for each chain. The table is sorted by the tail of the chains to make fast lookups possible.

5.2 Coverage and Costs

The time for the generation of the table as well as the storage and the lookup time and success probability for the attack is controlled by four parameters:

t	length of the rainbow chains
m	number of chains in a table
N	total number of distinct keys (2^{48} for DVB-CSA)
T	number of tables

The coverage of a single table is given by [10]:

$$R(t, m, N) := \prod_{i=1}^{t} \left(1 - \frac{m_i}{N}\right)$$

with $m_1 = m$ and

$$m_{i+1} = N(1 - e^{-\frac{m_i}{N}})$$

For T equivalent tables with different reduction functions we get:

$$C(t, m, N, T) := 1 - (1 - R(t, m, N))^T$$

as coverage of all tables combined.

The computational costs for a look-up in these tables is given by:

$$\sum_{i=1}^{t} T * i = \frac{T * t * (t + 1)}{2}$$

and the number of accesses to external memory (i.e. seeks on a HDD) is $T * t$.

To compute reasonable values for these parameters, we need an (efficient) implementation of f first, so that we know how many calls to f can be made during the attack.

[2] 1000_{10} in little-endian is $e8\ 03$.

6 Implementation

For our attack, we need to evaluate many instances of the one-way function. It is therefore desirable to either evaluate the one-way function in a very fast manner or to compute multiple instances of the function at once.

For the evaluation of our one-way function we need to do 1288 invocations (56 rounds $\times$ 23) of the DVB-CSA block cipher round function. As mentioned before, the round function consists of 8 bit XORs and two 8 bit mappings S and P. On most modern computer architectures, computations are carried out on larger bit vectors (mostly 32 or 64 bit). Also, sometimes there are vector instruction sets that allow to use even wider vectors, e.g. 128 or 256 bit.

We use a bitsliced implementation of the DVB-CSA block cipher encryption (except for the S-Box lookup). We do not use a lookup table for the permutation used in the key schedule but an bitsliced implementation of it. Many parts of our implementation are inspired by *libdvbcsa*[3], an open-source implementation of the DVB-CSA cipher. However, the current version of *libdvbcsa* only supports parallel encryption with the same key, but not with many different keys, as required for our precomputations. Also, our key schedule has a lower memory footprint, making it more suitable for GPUs.

6.1 SSE

SSE2[4] allows to do operations on 128 bit registers and thereby allows us to do e.g. 16 8 bit xor operations at the same time. Of course SSE2 has no implementation of either S nor P so they have to be implemented in some other way. Since P merely permutes the bit indices of a byte there is a very simple way to compute many instances of it at once on larger bit vectors. The developers of libdvbcsa[5] already implemented this in a very efficient way. Now for S it is a lot harder to find an algorithmic description of the function that is faster then an lookup table in memory. We decided to expand the 8 bit permutation S to an 16 bit permutation

$$S' : x||y \rightarrow S(x)||S(y)$$

and store precomputed values for it in a lookup table. This way we only need to do 8 instead of 16 lookups for 16 parallel computations of the block round function. The 16 bit lookup table does not necessarily fit into the L1 cache of a standard processor but will most likely fit into the L2 cache.

6.2 OpenCL

Modern Graphic Accelerators (GPU) are made up of many (hundreds) independent computing units that can be used to run the same code on different data. These so called compute units provide only limited amounts of fast memory/cache suited for lookup tables. We chose to implement the DVB-CSA block

[3] http://www.videolan.org/developers/libdvbcsa.html
[4] Streaming SIMD Extensions 2.

cipher here with a version of the permutation code that was originally written by the developers of *libdvbcsa* and a normal 8 bit lookup table for S. Our GPU implementation operates on 32 bit words and therefor computes 4 instances of the function per compute unit at the same time. The source code of our OpenCL kernel can be found in Appendix A.

7 Parameters

These high-speed implementations could be used to generate rainbow tables for various attack scenarios. Assuming that a hard-disk is able to perform 100 random accesses per second, and an adversary can encrypt about 4,000,000 cells on a GTX460 and about 500,000 cells on a single core CPU, we generated 3 parameter sets.

An adversary might be interested in recovering a DVB-CSA transmission in real-time. He needs to recover a single DVB-CSA key in less than 7 seconds. Using a GPU, the precomputations require 6 hard-disks and 6 TB of storage. Such a table can be precomputed on a single graphics card in less than 9 years. Using multiple graphics cards or faster graphics cards reduces the required time. If the key changes only every 10 seconds, the same tables can be stored on just 4 hard drives, without having to recompute them.

Alternatively, an adversary might not be interested in decoding a transmission in real time, or he would like to recover a static key from a station that only changes the key manually. If a key should be recovered within 30 minutes, this can be done with 120GB of precomputations on a graphics card (less than 8 years of precomputations on a single graphics card) or 525GB of precomputations on a CPU (less than 5 years of precomputations on a graphics card).

Some sets of possible rainbow table parameters are based on the desired speed at which keys would be recovered. We aimed at about 90% coverage. One interesting application of this attack would be PayTV with rapidly changing keys. For the purpose of breaking long term keys, a slower recovery rate would suffice.

Table 2. Suggested parameters

Scenario	# Tables	# Chains per table	Chain-length	Coverage	Storage
GPU 7sec per key	2	2^{38}	2000	93.457%	6TB
GPU 30min per key	3	2^{32}	68410	91.953%	120GB
SSE 30min per key	18	$2^{31.542}$	10000	85.722%	525GB

8 Experimental Results

We computed a small rainbow table with chains of 2000 elements and $2^{32.9008}$ chains. The table is round about 100 GB in size and has approx 5.4 % coverage. We created 23419 random keys and searched for their corresponding one-way

function outputs in this table we found 2464 preimages of which 1057 were the actual preimage we had been looking for. This corresponds with our expected success proabability.

9 Attack Options

There are several variants of this attack. First of all the rainbow table generation can be optimized:

9.1 Rainbow Table Optimizations

For our largest suggested parameter set (*GPU 7sec per key*), there are 2^{39} chains in total, stored in two tables with 2^{38} chains per table. Accordingly, chains with 2^{38} different inputs need to be computed. We can choose to use a counter or a similar method to generate the head of these chains. Therefor, only 5 instead of 6 bytes are required to encode the head of the chains. This allows us to save 1 byte per chain and reduce the size of the rainbow table without any side effects.

Also, for a table filled with 2^{38} chains that is sorted by the tail of the chains two consecutive chains will differ in 10 bits on average. Using a variable length encoding for the tails here will allow us to store the tails of the chains in only 2 instead of 6 bytes, for most chains. This additionally reduces the size of the table.

9.2 Harddisk Seek Performance

Our parameter sets have been chosen in a way that they allow the recovery of a key in 7 seconds or less, if it can be recovered using the table. Even if an adversary wants to decode a video stream in near real time, this requirement can be relaxed.

Assuming that a single table with chain length 2000 and 2^{40} chains is used, the total coverage is 96%. If 10 keys should be recovered with this table, at most 20000 seeks need to be performed. However, the probability that more than 7765 seeks need to be performed is below 0.1%. As a result, one can use a much lower number of harddisks, if the computed table has a high coverage. For tables with a small coverage, unsuccessful searches are more common so that the average number of seeks is closer to the maximum number of seeks for a lookup.

10 Countermeasures

For our attack, we exploit two properties: The key space of DVB-CSA is very small (only 2^{48} keys) and there are full MPEG TS cells that repeat often. Several countermeasures against this attack are possible:

Only 2^{48} possible keys is definitely a too small key space for an encryption system that is used to protect sensitive data. If the two checksum bytes of the

key would be chosen freely, 2^{64} keys would be possible. That would slow down our attack by a factor of 2^{16} and render it impossible with today's hardware for an attacker with a small budget. However, we do not know if that would cause interoperability problems with receivers and other equipment that check the checksum bytes of the key, or with key distribution systems, that can only generate keys with a correct checksum. Therefore, we cannot recommend this solution until compatibility with existing hardware has been ensured.

Even 2^{64} possible keys are not sufficient to protect highly sensitive data for a long time. If really high security is required, DVB-CSA should be redesigned and extended to at least 2^{128} possible keys. As far as we know, DVB-CSAv3 has been designed with a key space of 128 bits, but the design of DVB-CSAv3 is not open, so that we cannot evaluate the cryptographic strength of this algorithm. To use DVB-CSAv3, all Conditional Access Modules (CAM) need to be updated. If DVB-CSA is implemented in hardware, what we assume, they even need to be replaced. We think that this solution cannot be used on the short term, but is a great long term solution.

As a short-term countermeasure, we suggest a solution that can be deployed by changing only the equipment used at the sender, and receivers do not need to be updated. Our attack is based on the fact that MPEG TS cells filled completely with zero-bytes repeat frequently. The MPEG TS header specifies separately for every cell whether it is encrypted or not, and which key is used. We think that an DVB-CSA encryption device should check all cells to be encrypted for all-zero cells. Such cells should be sent unencryptedly. As a result, the attacker will not get a single zero-filled encrypted cell and will not be able to launch the attack. Depending on the video codec used, one should also check if there are other common cell plaintexts, and send them in plaintext too.

If all these countermeasures are not possible, there is still another way to prevent the attack, if only a single or a small number of tables have been generated and are publicly available. A sender can generate a random key, and check if that key can be recovered using these tables. If so, it is not used and the procedure is repeated. As a result, all keys used by a sender cannot be recovered using the public tables, but probably with tables that are not available to the public.

11 Conclusion

This paper shows that DVB-CSA can be broken in real time using standard PC hardware, if precomputed tables are available. These precomputations can be performed on a standard PC in years. This makes DVB-CSA useless for any application where real confidentiality is required. DVB-CSA might still be used to protect digital content, where an adversary is not interested in attacks on the system that recover less than 99% of the payload, and *can not be used to produce pirated Smart Cards*. The attack can be prevented with small changes to the DVB-CSA encryption equipment without having to alter the receivers.

We would like to thank everybody contributing to this paper. This especially includes Academica Senica in Taipei, Taiwan, that provided hardware to compute parts of the rainbow table used in this paper.

References

1. DVB Common Scrambling Algorithm - Distribution Agreements. Technical report, ETSI (June 1996)
2. ETSI Technical Report 289 - Digital Video Broadcasting (DVB); Support for use of scrambling and Conditional Access (CA) within digital broadcasting systems. Technical report, ETSI (October 1996)
3. ETSI EN 300 421 - Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services. Technical report, ETSI (August 1997)
4. ETSI EN 300 429 - Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for cable systems. Technical report, ETSI (April 1998)
5. ETSI EN 300 744 - Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television. Technical report, ETSI (January 2009)
6. Kühn, G.J., et al.: System and apparatus for blockwise encryption/decryption of data. Technical report (August 1998)
7. Kühn, G.J., et al.: ETSI EN 301 192 - Digital Video Broadcasting (DVB); DVB specification for data broadcasting. Technical report (April 2008)
8. Kim, W.-H., Chen, K.-J., Cho, H.-S.: Design and implementation of MPEG-2/DVB scrambler unit and VLSI chip. IEEE Transactions on Consumer Electronics 43(3), 980–985 (1997)
9. Li, W.: Security Analysis of DVB Common Scrambling Algorithm. In: Data, Privacy, and E-Commerce, ISDPE 2007, pp. 271–273. IEEE (2007)
10. Oechslin, P.: Making a Faster Cryptanalytic Time-Memory Trade-Off. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 617–630. Springer, Heidelberg (2003)
11. I. Rec. H. 262– iso/iec 13818-2. Information technology–Generic coding of moving pictures and associated audio information–Video (2000)
12. Simpson, L., Henricksen, M., Yap, W.-S.: Improved Cryptanalysis of the Common Scrambling Algorithm Stream Cipher. In: Boyd, C., González Nieto, J. (eds.) ACISP 2009. LNCS, vol. 5594, pp. 108–121. Springer, Heidelberg (2009)
13. Weinmann, R.-P., Wirt, K.: Analysis of the DVB Common Scrambling Algorithm (2003)
14. Wirt, K.: Fault Attack on the DVB Common Scrambling Algorithm. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganá, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) ICCSA 2005. LNCS, vol. 3481, pp. 577–584. Springer, Heidelberg (2005)

A OpenCL Kernel

```
#define u8  unsigned char
#define u32 unsigned int
#define uint64_t ulong
#define TID tid
#define CHAINLEN chainlen

__constant uchar sbox[256] =
  {
     0x3a, 0xea, 0x68, 0xfe, 0x33, 0xe9, 0x88, 0x1a, 0x83, 0xcf, 0xe1, 0x7f, 0xba, 0xe2, 0x38, 0x12,
     0xe8, 0x27, 0x61, 0x95, 0x0c, 0x36, 0xe5, 0x70, 0xa2, 0x06, 0x82, 0x7c, 0x17, 0xa3, 0x26, 0x49,
     0xbe, 0x7a, 0x6d, 0x47, 0xc1, 0x51, 0x8f, 0xf3, 0xcc, 0x5b, 0x67, 0xbd, 0xcd, 0x18, 0x08, 0xc9,
     0xff, 0x69, 0xef, 0x03, 0x4e, 0x48, 0x4a, 0x84, 0x3f, 0xb4, 0x10, 0x04, 0xdc, 0xf5, 0x5c, 0xc6,
     0x16, 0xab, 0xac, 0x4c, 0xf1, 0x6a, 0x2f, 0x3c, 0x3b, 0xd4, 0xd5, 0x94, 0xd0, 0xc4, 0x63, 0x62,
     0x71, 0xa1, 0xf9, 0x4f, 0x2e, 0xaa, 0xc5, 0x56, 0xe3, 0x39, 0x93, 0xce, 0x65, 0x64, 0xe4, 0x58,
     0x6c, 0x19, 0x42, 0x79, 0xdd, 0xee, 0x96, 0xf6, 0x8a, 0xec, 0x1e, 0x85, 0x53, 0x45, 0xde, 0xbb,
     0x7e, 0x0a, 0x9a, 0x13, 0x2a, 0x9d, 0xc2, 0x5e, 0x5a, 0x1f, 0x32, 0x35, 0x9c, 0xa8, 0x73, 0x30,
     0x29, 0x3d, 0xe7, 0x92, 0x87, 0x1b, 0x2b, 0x4b, 0xa5, 0x57, 0x97, 0x40, 0x15, 0xe6, 0xbc, 0x0e,
     0xeb, 0xc3, 0x34, 0x2d, 0xb8, 0x44, 0x25, 0xa4, 0x1c, 0xc7, 0x23, 0xed, 0x90, 0x6e, 0x50, 0x00,
     0x99, 0x9e, 0x4d, 0xd9, 0xda, 0x8d, 0x6f, 0x5f, 0x3e, 0xd7, 0x21, 0x74, 0x86, 0xdf, 0x6b, 0x05,
     0x8e, 0x5d, 0x37, 0x11, 0xd2, 0x28, 0x75, 0xd6, 0xa7, 0x77, 0x24, 0xbf, 0xf0, 0xb0, 0x02, 0xb7,
     0xf8, 0xfc, 0x81, 0x09, 0xb1, 0x01, 0x76, 0x91, 0x7d, 0x0f, 0xc8, 0xa0, 0xf2, 0xcb, 0x78, 0x60,
     0xd1, 0xf7, 0xe0, 0xb5, 0x98, 0x22, 0xb3, 0x20, 0x1d, 0xa6, 0xdb, 0x7b, 0x59, 0x9f, 0xae, 0x31,
     0xfb, 0xd3, 0xb6, 0xca, 0x43, 0x72, 0x07, 0xf4, 0xd8, 0x41, 0x14, 0x55, 0x0d, 0x54, 0x8b, 0xb9,
     0xad, 0x46, 0x0b, 0xaf, 0x80, 0x52, 0x2c, 0xfa, 0x8c, 0x89, 0x66, 0xfd, 0xb2, 0xa9, 0x9b, 0xc0,
  };

inline uint64_t rol64(uint64_t word, unsigned int shift)
{
  return (word << shift) | (word >> (64 - shift));
}

uint64_t permute_block(uint64_t k) {
        uint64_t n = 0;
        n  = rol64(k & 0x0080000002000000UL, 5);
        n ^= rol64(k & 0x0000000404000000UL, 6);
        n ^= rol64(k & 0x0000000080000000UL, 7);
        n ^= rol64(k & 0x0800000000000820UL, 10);
        n ^= rol64(k & 0x0000000000020000UL, 12);
        n ^= rol64(k & 0x1040000000108000UL, 13);
        n ^= rol64(k & 0x0000008000200000UL, 14);
        n ^= rol64(k & 0x0200002000000080UL, 15);
        n ^= rol64(k & 0x0004000000000000UL, 16);
        n ^= rol64(k & 0x0000020000000000UL, 18);
        n ^= rol64(k & 0x0000200000000001UL, 19);
        n ^= rol64(k & 0x0000000001000000UL, 23);
        n ^= rol64(k & 0x8000000000000000UL, 25);
        n ^= rol64(k & 0x0008000000000002UL, 26);
        n ^= rol64(k & 0x0002000000000400UL, 29);
        n ^= rol64(k & 0x0000000100000040UL, 30);
        n ^= rol64(k & 0x0000000000040000UL, 33);
        n ^= rol64(k & 0x0000000008004000UL, 36);
        n ^= rol64(k & 0x0000080040000000UL, 38);
        n ^= rol64(k & 0x0400001000000000UL, 40);
        n ^= rol64(k & 0x0000000000001000UL, 42);
        n ^= rol64(k & 0x0000400000000008UL, 43);
        n ^= rol64(k & 0x2000000000002000UL, 45);
        n ^= rol64(k & 0x0000000030000000UL, 46);
        n ^= rol64(k & 0x0000010800000000UL, 47);
        n ^= rol64(k & 0x0000000000000100UL, 48);
        n ^= rol64(k & 0x0000100000080000UL, 51);
        n ^= rol64(k & 0x0000000000000200UL, 52);
        n ^= rol64(k & 0x0000000000000004UL, 53);
        n ^= rol64(k & 0x0000000000010000UL, 55);
        n ^= rol64(k & 0x0110000200800000UL, 57);
        n ^= rol64(k & 0x4020000000000000UL, 59);
        n ^= rol64(k & 0x0001800000000000UL, 60);
        n ^= rol64(k & 0x0000000000000010UL, 61);
        n ^= rol64(k & 0x0000000000400000UL, 62);
        n ^= rol64(k & 0x0000044000000000UL, 63);
        return n;
}

#define I(i) (i*0x0101010101010101UL)

```

```
75  //4*6byte in ... 4 * 56 byte out
76  void keyschedule(const __private uchar *in, __private uint *out) {
77          ulong ks[7];
78          uint i,j;
79          for (i = 0; i < 4; i++) {
80                  ks[6] =                 in[3+(i*6)]+in[4+(i*6)]+in[5+(i*6)]; // checksum
81                  ks[6] = (ks[6] << 8) ^ in[5+(i*6)];
82                  ks[6] = (ks[6] << 8) ^ in[4+(i*6)];
83                  ks[6] = (ks[6] << 8) ^ in[3+(i*6)];
84                  ks[6] = (ks[6] << 8) ^ ((in[0+(i*6)]+in[1+(i*6)]+in[2+(i*6)])&0xff); // checksum
85                  ks[6] = (ks[6] << 8) ^ in[2+(i*6)];
86                  ks[6] = (ks[6] << 8) ^ in[1+(i*6)];
87                  ks[6] = (ks[6] << 8) ^ in[0+(i*6)];
88
89                  for (j = 6; j > 0; j--) {
90                          ks[j-1] = permute_block(ks[j]);
91                  }
92                  for (j = 0; j < 7; j++) {
93                          ks[j] ^= I(j);
94                  }
95                  for(j = 0; j < 56; j++) {
96                          out[j] = (out[j]<<8) ^ ((ks[j/8]>>(8*(j%8))) & 0xff);// << (24 - i*8);
97                  }
98          }
99  }
100
101 // csa roundfunction for any implementation of SBOX and P
102 #define RX(w0,w1,w2,w3,w4,w5,w6,w7,t0,t1,k){    \
103         t1 = SBOX(w7 ^ k);                      \
104         t0 = w1;                                \
105         w1 = w0 ^ w2;                           \
106         w2 = w0 ^ w3;                           \
107         w3 = w0 ^ w4;                           \
108         w4 = w5;                                \
109         w5 = w6 ^ P(t1);                        \
110         w6 = w7;                                \
111         w7 = w0 ^ t1;                           \
112         w0 = t0;}                               \
113
114 //lets keep it simple for the first shot
115
116
117 #define BS_BATCH_SIZE 32
118 #define BS_BATCH_BYTES 4
119
120 #define BS_VAL(n)       ((uint)(n))
121 #define BS_VAL32(n)     BS_VAL(0x##n)
122 #define BS_VAL16(n)     BS_VAL32(n##n)
123 #define BS_VAL8(n)      BS_VAL16(n##n)
124
125 #define BS_AND(a, b)    ((a) & (b))
126 #define BS_OR(a, b)     ((a) | (b))
127 #define BS_XOR(a, b)    ((a) ^ (b))
128 #define BS_XOREQ(a, b)  ((a) ^= (b))
129 #define BS_NOT(a)       (~(a))
130
131 #define BS_SHL(a, n)    ((a) << (n))
132 #define BS_SHR(a, n)    ((a) >> (n))
133 #define BS_SHL8(a, n)   ((a) << (8 * (n)))
134 #define BS_SHR8(a, n)   ((a) >> (8 * (n)))
135 #define BS_EXTRACT8(a, n) ((a) >> (8 * (n)))
136
137 #define BS_EMPTY()
138
139 inline uint bsperm(uint sbox_out) {
140         return BS_OR(
141                         BS_OR(
142                               BS_OR (BS_SHL (BS_AND (sbox_out, BS_VAL8(29)), 1),
143                                      BS_SHL (BS_AND (sbox_out, BS_VAL8(02)), 6)),
144                               BS_OR (BS_SHL (BS_AND (sbox_out, BS_VAL8(04)), 3),
145                                      BS_SHR (BS_AND (sbox_out, BS_VAL8(10)), 2))),
146                         BS_OR(       BS_SHR (BS_AND (sbox_out, BS_VAL8(40)), 6),
147                                      BS_SHR (BS_AND (sbox_out, BS_VAL8(80)), 4)));
148 }
149
150
151 #define P(x) bsperm(x)
152 #define SBOX(x) lookup(x,sbox)
```

```
153
154  uint lookup(uint x, __constant uchar *lut) {
155          uint r = 0;
156          r = lut[x&0xff];x>>=8;
157          r ^= lut[x&0xff]<<(8);x>>=8;
158          r ^= lut[x&0xff]<<(16);x>>=8;
159          r ^= lut[x&0xff]<<(24);x>>=8;
160          return r;
161  }
162
163  void encrypt(__private const uint *key, __private uint *i) {
164          uint t1,t2,k;
165          uint w0,w1,w2,w3,w4,w5,w6,w7;
166
167          w0 = i[0];
168          w1 = i[1];
169          w2 = i[2];
170          w3 = i[3];
171          w4 = i[4];
172          w5 = i[5];
173          w6 = i[6];
174          w7 = i[7];
175
176          int r;
177          for (r = 0; r < 56; r++){
178                  k = key[r];
179                  RX(w0,w1,w2,w3,w4,w5,w6,w7,t1,t2,k)
180          }
181
182
183          i[0] = w0;
184          i[1] = w1;
185          i[2] = w2;
186          i[3] = w3;
187          i[4] = w4;
188          i[5] = w5;
189          i[6] = w6;
190          i[7] = w7;
191  }
192
193
194  #define expand(x) {x^=(x<<8);x^=(x<<16);}
195  void r(uint tid, uint idx, __private uint *in) {
196          int i;
197          for (i = 0; i < 4; i++) {
198                  uint x = idx&0xff;
199                  expand(x)
200                  in[i] ^= x;
201                  idx>>=8;
202          }
203          for (i = 0; i < 2; i++) {
204                  uint x = tid&0xff;
205                  expand(x)
206                  in[i+4] ^= x;
207                  tid>>=8;
208          }
209  }
210
211
212  __kernel void csa (__global const u8* input,
213                  __global u8* output,
214                  const int tid,
215                  const int chainlen,
216                  const int num)
217  {
218          const int idx = get_global_id(0);
219          if (idx > num) return;
220          int i;
221          __private u8 k[24];
222          for (i = 0; i < 24; i++) {
223                  k[i] = input[(24*idx)+i];
224          }
225          __private uint key[56];
226          __private uint zero[8];
227          int j,l;
228          for (j = 0; j < CHAINLEN; j++){
229                  keyschedule(k,key);
230                  for(i = 0; i < 8; i++)
```

```
231                         zero[i] = 0;
232                 for (i = 0; i < 23; i++){
233                         encrypt(key,zero);
234                 }
235                 r(TID,j,zero);
236                 for (i = 0; i < 4; i++) {
237                         for (l = 0; l < 6; l++)
238                                 k[i*6+l] = (zero[l]>>(24-8*i));
239                 }
240         }
241         for (i = 0; i < 24; i++)
242                 output[24*idx+i] = k[i];
243 }
```

On the Role of Expander Graphs in Key Predistribution Schemes for Wireless Sensor Networks

Michelle Kendall and Keith M. Martin

Information Security Group, Royal Holloway,
University of London, Egham, Surrey, TW20 0EX, UK

Abstract. Providing security for a wireless sensor network composed of small sensor nodes with limited battery power and memory can be a non-trivial task. A variety of key predistribution schemes have been proposed which allocate symmetric keys to the sensor nodes before deployment. In this paper we examine the role of expander graphs in key predistribution schemes for wireless sensor networks. Roughly speaking, a graph has good expansion if every 'small' subset of vertices has a 'large' neighbourhood, and intuitively, expansion is a desirable property for graphs of networks. It has been claimed that good expansion in the product graph is necessary for 'optimal' networks. We demonstrate flaws in this claim, argue instead that good expansion is desirable in the intersection graph, and discuss how this can be achieved. We then consider key predistribution schemes based on expander graph constructions and compare them to other schemes in the literature. Finally, we propose the use of expansion and other graph-theoretical techniques as metrics for assessing key predistribution schemes and their resulting wireless sensor networks.

Keywords: Wireless Sensor Networks, Key Predistribution, Expander Graphs.

1 Introduction

A *wireless sensor network* (WSN) is a collection of small, battery powered devices called sensor nodes. The nodes communicate with each other wirelessly and the resulting network is usually used for monitoring an environment by gathering local data such as temperature, light or motion. As the nodes are lightweight and battery powered, it is important to consider battery conservation in order to allow the network to remain effective for the appropriate period of time, and to ensure that the storage required of the nodes is not beyond their memory capacity.

WSNs are suitable for deployment in many different environments, including potentially hostile areas such as military or earthquake zones, where it would be dangerous or impractical to carry out the monitoring of data gathering by hand. In hostile environments it may be necessary to encrypt messages for security and / or authentication. Various cryptographic key management schemes

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 62–82, 2012.

have been proposed for such scenarios. In some cases there is an online key server or base station to distribute keys to the nodes where needed; if not, *key predistribution schemes* (KPSs) are required, which assign keys to nodes before deployment. Due to the resource-constrained nature of the nodes, it may be infeasible to use asymmetric cryptographic techinques in some WSN scenarios, and so we consider symmetric key predistribution schemes.

Since networks may be modelled as graphs, tools from graph theory have been used both to analyse and to design networks. In particular, we explore the role of expander graphs in KPSs for WSNs. The expansion of a graph is a measure of how well connected it is, and how difficult it is to separate subsets of vertices; we will see the precise definition in Sect. 2.3. The term 'expander graphs' is used informally to refer to graphs with good expansion.

In 2006, expander graph theory was introduced to the study of KPSs for WSNs from two perspectives. On the one hand, Camtepe et al [5] showed that a mathematical construction for an expander graph could be used to design a KPS, resulting in a network which is well connected under certain constraints. On the other hand, Ghosh [13] claimed that good expansion is a necessary condition for 'optimal' WSNs. We examine these claims and determine the role of expander graphs in KPSs for WSNs.

Firstly, we show that Ghosh's claim is flawed but identify where expansion properties are desirable for WSNs, namely in the intersection graph rather than the product graph. We then analyse the effectiveness of constructions for KPSs based on expander graphs, showing that they provide perfect resilience against an adversary but lower connectivity and expansion than many existing KPSs, for the same network size and key storage. We argue that expansion is an important metric for assessing KPSs to be used alongside the other common metrics of key storage, connectivity and resilience for a given network size. However, we note the difficulty of finding the expansion coefficient of a graph and so propose estimating the expansion and using other graph-theoretical techniques to indicate weaknesses.

We begin by introducing the relevant terminology and concepts in Sect. 2. In Sect. 3 we outline Ghosh's claims and show by means of a counter-example that his conclusion is misdirected towards expansion in product graphs rather than intersection graphs. In Sect. 4 we discuss how to maximise the probability of a high expansion coefficent in the intersection graph, and in Sect. 5 we analyse the extent to which KPSs based on expander graph constructions achieve this, in comparison to other schemes from the literature. Finally, in Sect. 6 we suggest practical metrics for analysing and improving KPSs and the resulting WSNs, and conclude in Sect. 7.

2 Background

2.1 Key Predistribution Schemes for Wireless Sensor Networks

A *key predistribution scheme* is a well-defined method for determining the combination of keys which should be stored on each node before deployment. Once

the nodes have been deployed in the environment, they broadcast identifiers which uniquely correspond to the keys they store, and determine the other nodes (within communication range) with which they share at least one common key, in order to form a WSN.

There are many ways of designing a KPS, and different KPSs suit different WSN applications. We consider KPSs which assign symmetric keys, since small sensor nodes are resource-constrained with low storage, communication and computational abilities, and are often unable to support asymmetric cryptography. In order to make best use of the nodes' limited resources, it is usually desirable to minimise the *key storage* requirement whilst maximising the *connectivity* and *resilience* of a network of n nodes. We now define these concepts more precisely.

- *Key storage* is the maximum number of keys which an individual node is required to store for a particular KPS.
- *Connectivity* of a network can be measured or estimated both *globally* and *locally*. We will refer again to global connectivity in Sect. 6 but in general we will use the measure of local connectivity Pr_1. This is defined to be the probability that two randomly-chosen nodes are 'connected' because they have at least q keys in common. Most KPSs require nodes to share just one key before they can establish a secure connection, ie. $q = 1$, and so Pr_1 is simply the probability that a random pair of nodes have at least one key in common. Some schemes such as the q-composite scheme of Chan et al. [6] introduce a threshold such that nodes may only communicate if they have at least $q > 1$ common keys. Where two nodes share more than q keys, some protocols dictate that they should use a combination of those keys, such as a hash, to encrypt their communications.
- *Resilience* is a measure of the network's ability to withstand damage from an adversary. We use the adversary model of a continuous, listening adversary, which can listen to any communication across the network and continually over time 'compromise' nodes, learning the keys which they store. We measure the resilience with the parameter fail_s: we suppose that an adversary has compromised s nodes, and then compute the probability that a link between two uncompromised nodes in the network is compromised, that is, the adversary knows the key(s) being used to secure it. Equivalently, fail_s measures the fraction of compromised links between uncompromised nodes throughout the network, after an adversary has compromised s nodes. Notice that high resilience corresponds to a low value of fail_s. We say that a network has *perfect resilience* against such an adversary if $\mathsf{fail}_s = 0$ for all $1 \leq s \leq n - 2$.

To illustrate the trade-offs required between these three parameters, we consider some trivial examples of KPSs.

1. *Every node is assigned a single key k before deployment.*
   This would require minimal key storage and ensure that any pair of nodes could communicate securely, so $\mathsf{Pr}_1 = 1$ for all pairs of nodes. However, there would be minimal resilience against an adversary, as the compromise

of a single node would reveal the key k, rendering all other links insecure. Formally, $\mathsf{fail}_s = 1$ for all $1 \leq s \leq n-2$.

2. *A unique key is assigned to every pair of nodes.*
That is, for all $1 \leq i, j \leq n$, nodes n_i and n_j are both preloaded with a key k_{ij}, where $k_{ij} \neq k_{lm}$ for all pairs $(l, m) \neq (i, j)$. This is called the *complete pairwise* KPS. Such a KPS would have perfect resilience and maximum connectivity, as $\mathsf{Pr}_1 = 1$ for all pairs of nodes. However, each node would have to store $n-1$ keys, which is infeasible when n is large.
3. *Every node is assigned a single unique key.*
Whilst providing minimal key storage and perfect resilience, this KPS has no connectivity, as $\mathsf{Pr}_1 = 0$ for all pairs of nodes.

We see, then, that it is trivial to optimise any two of these three parameters. However, for many WSN applications these schemes are inappropriate, and so we consider KPSs which find trade-offs between all three of these metrics. A variety of such KPSs have been proposed, both deterministic and random, a survey of which is given in [4] [7] [19] [23].

We now describe the Eschenauer Gligor KPS [12] as an example of a KPS which provides values for the three metrics which are appropriate for many WSN scenarios. It will also be used as a comparison for KPSs based on expander graph constructions in Sect. 5.

Example 1. The Eschenauer Gligor KPS [12] works in the following way. A key pool $\mathcal{K}$ is generated. To each node n_i is assigned a random subset of k keys from $\mathcal{K}$. The probability of two nodes sharing at least one key is

$$\mathsf{Pr}_1 = 1 - \frac{\binom{|\mathcal{K}|-k}{k}}{\binom{|\mathcal{K}|}{k}} \quad . \tag{1}$$

An equivalent formula is given in [12]. We verify (1) by considering the probability of two arbitrary nodes n_i and n_j sharing no common keys. If node n_i stores a set S_i of k keys, node n_j stores S_j and $S_i \cap S_j = \emptyset$, then every key of S_j must have been picked from the key pool $\mathcal{K} \setminus S_i$, that is from a set of $|\mathcal{K}| - k$ keys. Therefore the probability of two nodes having no keys in common is equal to the number of ways of choosing k keys from a key pool of $|\mathcal{K}| - k$, divided by the number of ways of choosing k keys from the full key pool.

As explained in [6], the resilience after the compromise of s nodes is

$$\mathsf{fail}_s = \left(1 - \left(1 - \frac{k}{|\mathcal{K}|}\right)^s\right)^q \quad . \tag{2}$$

where q is the number of keys shared between two randomly-chosen uncompromised nodes. This leads to the intuitive result that if the adversary has compromised one node, learning k keys, and two randomly-chosen uncompromised nodes share $q = 1$ key k_1, then the probability of the adversary knowing k_1 is $\mathsf{fail}_1 = \frac{k}{|\mathcal{K}|}$. If $q > 1$ keys are shared between the nodes then the value of fail_1 will be smaller, as $\mathsf{fail}_1 = \left(\frac{k}{|\mathcal{K}|}\right)^q$.

Table 1 demonstrates the value of Pr_1 *and upper bound for* fail_1 *for some different sizes of key pool and key storage. (We assume for this example that all nodes are within communication range of one another.) It can be seen that by adjusting the values of* $|\mathcal{K}|$ *and* k*, with* k *small, we can achieve arbitrarily large* Pr_1 *whilst keeping* fail_1 *relatively small. This makes the Eschenauer Gligor KPS appropriate for many WSN scenarios.*

Table 1. Example values for an Eschenauer Gligor KPS

$\lvert\mathcal{K}\rvert$	k	Pr_1	fail_1
500	25	0.731529	0.050
500	50	0.996154	0.100
1000	25	0.473112	0.025
1000	50	0.928023	0.050

In accordance with most papers in the literature, we will use key storage, connectivity and resilience along with network size as metrics for comparing KPSs. Network size is relevant since for small networks the complete pairwise KPS is practical, and because some KPSs are not adaptable for all sizes of network. For example we will see in Sect. 5 that the KPS proposed by Camtepe et al [5] is only possible for networks of size $n = t + 1$ where t is a prime congruent to 1 mod 4.

Later, in Sect. 6, we will propose that in addition to network size, key storage, connectivity and resilience, it is important to consider expansion as a metric when comparing KPSs.

2.2 Graph Theory

We now introduce some graph-theoretic definitions, beginning with general terminology in this section before giving the specific definitions related to expander graphs in Sect. 2.3.

A *graph* $G = (V, E)$ is a set of *vertices* $V = \{v_1, \ldots, v_n\}$ and a set of edges E. We use the notation $(v_i, v_j) \in E$ to express that there is an edge between the vertices v_i and v_j, and we say that the edge (v_i, v_j) is *incident* to its endpoints v_i and v_j. Wherever an edge (v_i, v_j) exists, v_i and v_j can be said to be *adjacent*.

All graphs considered in this paper will be *simple graphs*, that is, they are *unweighted*, *undirected* and do not contain *self-loops* or *multiple edges*. These terms respectively mean that vertices are not assigned different weights, edges are not directed from one vertex to the other, there are no edges from the a node to itself, and any edge between two vertices is unique.

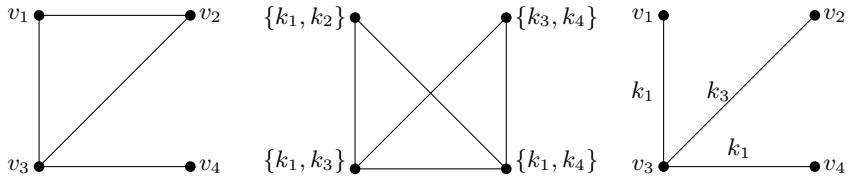
Given subsets of vertices $X, Y \subset V$, the set of edges which connect X and Y is denoted

$$E(X, Y) = \{(x, y) : x \in X, y \in Y \text{ and } (x, y) \in E\} \ ,$$

and the *complement* $\overline{X}$ of X is the vertices which are not in X, that is, $\overline{X} = V \backslash X$.

An ordered set of consecutive edges $\{(v_{i1}, v_{i2}), (v_{i2}, v_{i3}), \ldots, (v_{i(p-1)}, v_{ip})\}$ in which all the vertices $v_{i1}, v_{i2}, \ldots, v_{ip}$ are distinct is called a *path* of length $p-1$. A *cycle* is a 'closed' path which begins and ends at the same vertex, ie. a cycle is a path $\{(v_{i1}, v_{i2}), (v_{i2}, v_{i3}), \ldots, (v_{i(p-1)}, v_{ip})\}$ where $v_{i1}, v_{i2}, \ldots, v_{i(p-1)}$ are distinct but $v_{i1} = v_{ip}$. We say that a graph is *connected* if there is a path between every pair of vertices, and *complete* if there is an edge between every pair of vertices. The *degree* $d(v)$ of a vertex v is the number of edges incident to that vertex. If all nodes have the same degree r, we call the graph *r-regular*.

We draw a graph of a WSN by representing the nodes as vertices and the 'connections' as edges. To be precise in our analysis, we distinguish between the two possible types of 'connection' and consider the separate constituent graphs of a network: the *communication graph* $G_1 = (V, E_1)$ where $(v_i, v_j) \in E_1$ if v_i and v_j are within communication range, and the *key graph* $G_2 = (V, E_2)$ where $(v_i, v_j) \in E_2$ if v_i and v_j share at least q common keys. An example of a communication graph and a key graph are given in Fig. 1.



(a) Comm. graph (b) Key graph (c) Intersection graph

Fig. 1. Example of corresponding communication, key and intersection graphs

If the communication graph is complete, it is often omitted from the analysis as there is no need to check whether nodes can communicate. However, as we will explain in Sect. 4, the communication graph is commonly modelled using a random graph, and it then becomes important to analyse how the communication and key graphs relate to each other.

We say that two nodes v_i and v_j can *communicate securely* if $(v_i, v_j) \in E_1 \cap E_2$, that is if they are adjacent in the *intersection* graph $G_1 \cap G_2 = (V, E_1 \cap E_2)$. This is illustrated in Fig. 1(c). We note that the standard definition of an intersection graph is $G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$, but throughout this paper $V_1 = V_2$ and so we simply refer to the set of vertices as V.

If two nodes are not adjacent in the intersection graph then there are usually ways for them to communicate by routing messages through intermediary nodes and/or establishing a new key. Since any protocol for either of these methods requires extra communication overheads, it is desirable to minimise the *diameter* of the intersection graph, that is to minimise the longest path length between nodes. Similarly, it may also be desirable to minimise the average path length of the intersection graph.

Finally, we introduce another way of combining two graphs, which will be needed for Sect. 3 where we consider Ghosh's claims.

Definition 1. *The (Cartesian)* product graph *of two graphs* $G = (V_G, E_G)$ *and* $H = (V_H, E_H)$ *is defined as* $G.H = (V_G \times V_H, E_{G.H})$*, where the set of edges* $E_{G.H}$ *is defined in the following way:* $(uv, u'v') \in E_{G.H}$ *if*

$$\begin{gathered}(u = u' \text{ or } (u, u') \in E_G) \\ \text{and} \\ (v = v' \text{ or } (v, v') \in E_H) \quad .\end{gathered}$$

We will now define expander graphs and explain why their properties are desirable for WSNs.

2.3 Expander Graphs

For a thorough survey of expander graphs and their applications, see [14]. The *expansion* of a graph is a measure of the quality of its connectivity.

Definition 2. *A finite graph* $G = (V, E)$ *is an* ϵ-expander graph*, where the edge-expansion coefficient* ϵ *is defined by*

$$\epsilon = \min_{S \subset V : |S| \leq \frac{|V|}{2}} \left(\frac{|E(S, \overline{S})|}{|S|} \right) \quad ,$$

where $|E(S, \overline{S})|$ *denotes the number of edges from the set* S *to its complement.*

The phrase 'expander graph' is used informally to refer to graphs with good expansion, that is, graphs with a high value of ϵ, as we explain below. We note that definitions vary across the literature, in particular some defintions use the strict inequality $|S| < \frac{|V|}{2}$. Another name for the edge-expansion coefficient is the *isoperimetric number*, and in a graph where every vertex has the same weight this is equivalent to the *Cheeger constant*; see [9] for further details.

We now explore what the definition of the edge-expansion coefficient ϵ means, and why a high value of ϵ is desirable, through the following observations:

- If $\epsilon = 0$ then we see from the definition that there exists a subset $S \subset V$ without any edges connecting it to the rest of the graph. This implies that the graph is not connected.
- A graph is connected if and only if $\epsilon > 0$ (see proof of Proposition 1), hence all connected graphs are ϵ-expander graphs for some positive value of ϵ.
- If ϵ is 'small', for example $\epsilon = \frac{1}{100}$, then there exists a set of vertices S which is only connected to the rest of the graph by one edge per 100 nodes in S. This is undesirable for a WSN for the following reasons:
  - The set S is vulnerable to being 'cut off' from the rest of the network by a small number of attacks or faults. If S contains $c \times 100$ nodes then there are only c edges between S and $\overline{S}$. A small number of compromises or failures amongst the particular $\leq 2c$ nodes incident to these edges will render all communication between S and $\overline{S}$ insecure.

  - Since S is connected to the rest of the network by comparatively few edges, a higher communication burden is placed on the small set of $\leq 2c$ nodes, since a higher proportion of data needs to be routed through them. This will drain the batteries of the nodes nearest to the edges between S and $\overline{S}$ faster than those of an average node, so that after some period of time they will run out of energy, disconnecting S from the rest of the network even though many nodes in S may still have battery power remaining.
  - Reliance on a small number of edges to connect large sets of nodes may create bottlenecks in the transmission of data through the network, making data collection and/or aggregation less efficient.
- If ϵ is larger, particularly if $\epsilon > 1$, then there is no 'easy' way to disconnect large sets of nodes, and communication burdens, battery usage and data flow are more evenly spread.

We see from these observations that intersection graphs with higher values of ϵ are more desirable for WSNs. A graph with a 'large' value of ϵ is often said to have 'good expansion'. The the size of ϵ is subject to the following bounds.

Proposition 1. *For any connected graph $G = (V, E)$ with $|V| \geq 2$,*

$$0 < \epsilon \leq \min_{v \in V} d(v) \ .$$

Proof. We begin by considering the lower bound. Suppose for a contradiction that $\epsilon = 0$. Then there exists a set $S \subset V$ such $|E(S, \overline{S})| = 0$. This contradicts the fact that G is connected. Since ϵ cannot be negative, we have that $\epsilon > 0$.

For the upper bound, consider the set $S = \{v\}$ where $v \in V$. It is clear that $|E(S, \overline{S})| = d(v)$, where $d(v)$ is the degree of v as defined in Sect. 2.2, and so $\frac{|E(S,\overline{S})|}{|S|} = \frac{d(v)}{1} = d(v)$. Since the definition of the edge-expansion coefficient ϵ uses the *minimum* value over all $S \subset V$ with $|S| \leq \frac{|V|}{2}$, we have that $\epsilon \leq \min_{v \in V} d(v)$. □

In addition to the observations made above, graphs with good expansion also have low diameter, logarithmic in the size of the network [14] and contain multiple short, disjoint paths between nodes [16], which is beneficial for schemes like the multipath reinforcement of Chan et al. [6]. These properties mean that key graphs with good expansion are particularly desirable for WSNs.

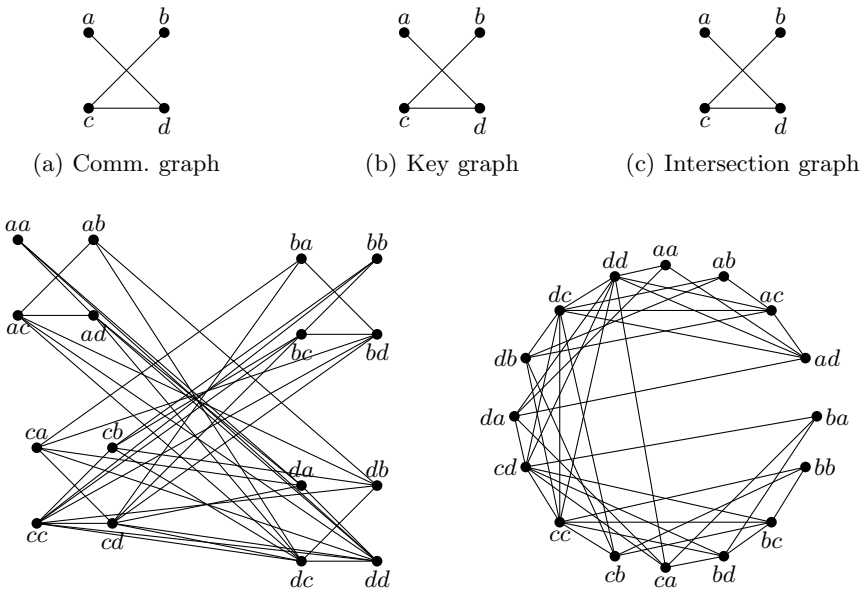
The papers by Camtepe et al. [5] and Shafiei et al. [21] propose KPSs based on expander graph constructions. These methods of designing a KPS ensure that the key graph has good expansion, and we further examine these proposals in Sect. 5. First, we consider the claims made by Ghosh in [13] about the necessity of good expansion for 'optimal' networks.

3 Expansion in Product Graphs

In [13] Ghosh considers KPSs with large network size, low key storage per node, high connectivity and high resilience. He considers jointly 'optimising' these

parameters, although exactly what this means is unclear, since different applications will prioritise them differently. Nevertheless, he argues that if a KPS is in some sense 'optimal', the product graph of the key graph and communication graph must have 'good expansion properties'. We show by a counterexample that expansion in the product graph is not a helpful measure because the product graph is almost inevitably an expander graph. Additionally, we show that the product graph is unable to capture the required detail to analyse a WSN, and that it is the intersection graph where such analysis is relevant.

In Fig.s 2 and 3 we consider examples of product graphs and examine how they relate to their constituent communication and key graphs. Figure 2 shows a communication and a key graph, and their corresponding intersection and product graphs. The product graph is represented in Fig. 2(d) in a way which demonstrates its construction, and redrawn in Fig. 2(e) for clarity.



(a) Comm. graph (b) Key graph (c) Intersection graph

(d) Cartesian product graph (e) Cartesian product graph re-drawn

Fig. 2. A product graph corresponding to an identical communication and key graph pair

Figure 2(d) illustrates that the product graph construction results in four copies of the key graph, connected to each other in a pattern which resembles the communication graph. To provide an alternative perspective, we re-draw the same graph with the vertices arranged in a circle in Fig. 2(e). To understand the construction, recall Def. 1 which defines the vertices and edges of the product graph. We find that there is an edge in the product graph $(ac, ab) \in E_{G.H}$ because

$a = a$ and $(c, b) \in E_H$. Similarly, $(ca, ba) \in E_{G.H}$ because $(c, b) \in E_G$ and $a = a$. However, we find that $(aa, ab) \notin E_{G.H}$ because whilst $a = a$, $(a, b) \notin E_H$.

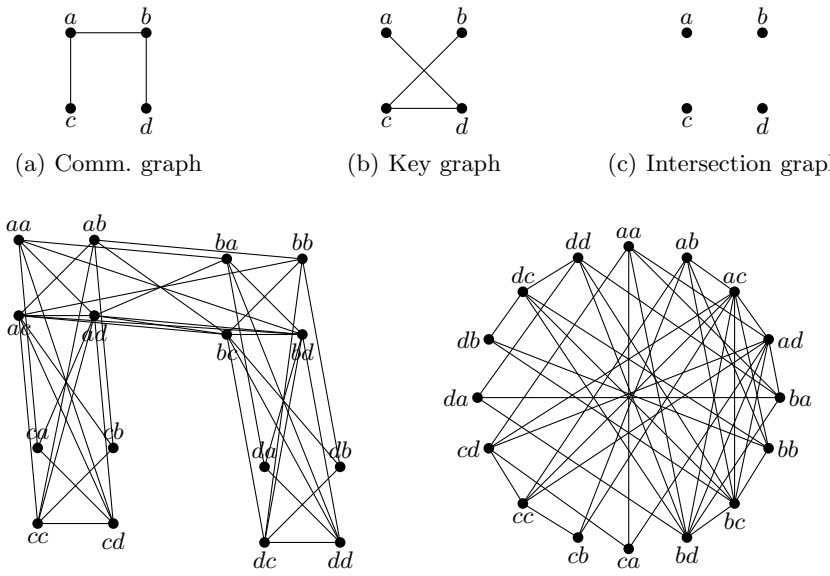
In Fig. 2 the communication and key graphs are identical, giving the best possible case for intersection. We now calculate the expansion coefficient of the product graph. Consider sets S of $1, 2, \ldots, 8$ vertices (recall from the definition that we should consider subsets S with $|S| \leq \frac{|V|}{2}$, and here $|V| = 16$). We observe that any single vertex is connected to the rest of the graph by at least three edges, any set of two vertices is connected to the rest of the graph by at least six edges, etc., so that

$$\epsilon = \min \left\{ \frac{3}{1}, \frac{6}{2}, \frac{9}{3}, \frac{9}{4}, \frac{11}{5}, \frac{16}{6}, \frac{12}{7}, \frac{10}{8} \right\} .$$

That is, $\epsilon = \frac{10}{8} = \frac{5}{4}$, so the product graph of Fig. 2 has expansion coefficient $\epsilon = \frac{5}{4}$.

Now consider Fig. 3, where we have the same key graph but the communication graph is altered. It has the same number of edges as in Fig. 2 but in such a way that the intersection graph, shown in Fig. 3(c), has no edges. Clearly for WSN purposes this would mean that no secure communication was possible.

However, the product graph does have edges, and indeed appears well connected. By observation, we find that it too has expansion coefficient $\epsilon = \frac{5}{4}$.



(a) Comm. graph (b) Key graph (c) Intersection graph

(d) Cartesian product graph (e) Cartesian product graph re-drawn

Fig. 3. A product graph corresponding to a communication and key graph pair with empty intersection

Indeed, after some inspection, we find that the product graphs of Fig.s 2 and 3 are isomorphic, using a simple bijection to relabel vertices as follows:

Fig. 2(e)		Fig. 3(e)
$(a*)$	$\rightarrow$	$(c*)$
$(b*)$	$\rightarrow$	$(d*)$
$(c*)$	$\rightarrow$	$(b*)$
$(d*)$	$\rightarrow$	$(a*)$

This means that all graph-theoretic properties of connectivity, expansion, degree, diameter etc. are identical between the two product graphs. From this we see that a product graph with good expansion can occur when the key and communication graphs intersect 'fully', ie. when $E_G \cap E_H = E_G = E_H$, and when there are no edges in the intersection, ie. $E_G \cap E_H = \emptyset$. This shows that the expansion of the product graph certainly does not correspond to any degree of 'optimality' regarding the intersection graph and therefore the WSN. In particular, it strongly suggests that expansion in the product graph is not a good tool for analysing the connectivity of WSNs without reference to the intersection graph. Ghosh's claim that an 'optimal' combination of key and communication graph will result in a product graph with good expansion tells us very little, since good expansion in the product graph is almost inevitable, as we will now explain.

Proposition 2. *A (Cartesian) product graph $G.H = (V_G \times V_H, E_{G.H})$ is connected if and only if both $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are connected.*

Proof. If the product graph is connected then there is a path between all pairs of vertices $u_1v_1, u_pv_q \in V \times V$, say

$$(u_1v_1, u_2, v_2), (u_2v_2, u_3v_3), \ldots, (u_{p-1}v_{q-1}, u_pv_q) \ .$$

Using the definition of the product graph, this implies that either $u_1 = u_2$ or $(u_1, u_2) \in E_G$, and indeed for all $1 \leq i \leq p-1$, either $u_i = u_{i+1}$ or $(u_i, u_{i+1}) \in E_G$. Thus either $u_1 = u_p$ or there is a path from u_1 to u_p in G. Since this is true for all pairs of vertices $u_1, u_p \in V$, we have that G is connected. By the same argument, H is also connected.

Suppose that G and H are both connected graphs. Then for each distinct pair of vertices $u_1, u_p \in V_G$, there is a path between them, say

$$(u_1, u_2), (u_2, u_3), \ldots, (u_{p-1}, u_p) \ .$$

Similarly, for each distinct pair of vertices $v_1, v_q \in V_H$, there is a path, say

$$(v_1, v_2), (v_2, v_3), \ldots, (v_{q-1}, v_q) \ .$$

By the definition of $E_{G.H}$, we have that $(u_1v_1, u_2v_2) \in E_{G.H}$. Thus we can construct a path

$$(u_1v_1, u_2v_2), (u_2v_2, u_3v_3), \ldots, (u_{p-1}v_{q-1}, u_pv_q)$$

in $G.H$ between any pair of vertices, and therefore $G.H$ is connected. □

Corollary 1. *If G and H are connected, the product graph $G.H$ has expansion coefficient $\epsilon_{G.H} > 0$.*

Proof. Recall from Prop. 1 that a connected graph is an expander graph for some value of ϵ. Therefore, if G and H are connected, the product graph will be an expander graph for some value of $\epsilon_{G.H} > 0$. □

We conjecture that with high probability, $\epsilon_{G.H} > \epsilon_G, \epsilon_H$ and $\epsilon_{G.H} >> 0$. We justify this by considering the comparatively large degrees of nodes in the product graph, and the product graph's similarity to an expander graph construction.

For any node $v \in V$ with degrees $d_G(v), d_H(v)$ in the communication and key graphs respectively, we can compute its degree in the product graph as

$$d_{G.H}(v) = d_G(v)d_H(v) + d_G(v) + d_H(v) \quad . \tag{3}$$

Using Prop. 1, we have that

$$\epsilon_{G.H} \leq \min_{v \in V} \left(d_G(v)d_H(v) + d_G(v) + d_H(v)\right) \quad ,$$

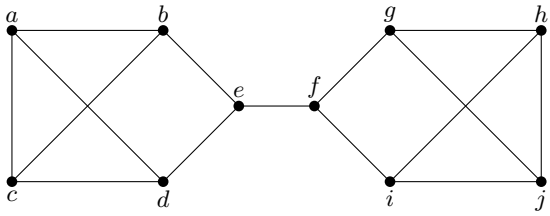
a much higher bound than for the constituent graphs. Since, on average, vertices of the product graph have higher degree than vertices in the constituent graphs, and since the construction of the product graph makes 'isolated' sets of vertices extremely unlikely, we see that $\epsilon_{G.H}$ is likely to be large, and in particular greater than either of ϵ_G and ϵ_H. By comparison, the expansion coefficient of the intersection graph $\epsilon_{G \cap H}$ is forced to be no more than those of the constituent graphs, ϵ_G and ϵ_H, as explained in the next section.

Additionally, the construction of the product graph is not dissimilar to that of the *zig-zag product* graph presented in [20] as an expander graph construction, and used by Shafiei et al in [21] to produce key graphs with good expansion. We see then that expansion in the product graph is inevitable if the constituent graphs are connected, is likely to be 'good', and does not imply anything about the quality of the connectivity or expansion in the intersection graph, where it is needed. Ghosh does not justify his choice of using the product graph as a means of studying two graphs simultaneously, and we conclude that there are no benefits to doing so. In order to capture the relevant interaction between the key and communication graphs, the intersection graph is the relevant tool, and it is in the intersection graph where good expansion is desired.

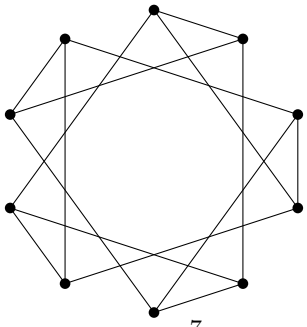
4 Expansion in Intersection Graphs

We claim that when comparing two WSNs of the same size with identical key storage, connectivity and resilience parameters, the WSN represented by the intersection graph with higher expansion will be the more robust, with a more evenly distributed flow of data. We justify this using the following example.

Example 2. Consider Fig. 4 and suppose that these are two intersection graphs, representing WSNs. Each graph is 3-regular on 10 nodes. We suppose that an



(a) $\epsilon = \frac{1}{5}$



(b) $\epsilon = \frac{7}{5}$

Fig. 4. Examples of 3-regular graphs on 10 nodes with different expansion parameters.

Eschenauer Gligor KPS (as described in Sect. 2.1) has been used to construct the key graph, where each node stores three keys chosen randomly from a pool of 25 keys. To simplify the analysis, we say that where nodes have more than one key in common, they select just one of them for use in securing their communications.

Using the formulae given in Sect. 2.1 we calculate that $\mathsf{Pr}_1 = 1 - \frac{\binom{22}{3}}{\binom{25}{3}} \approx 0.33$, $\mathsf{fail}_1 = \frac{3}{25}$ *and* $\mathsf{fail}_s = 1 - \left(1 - \frac{3}{25}\right)^s$ *for both graphs. Calculating the expansion, we observe that in Fig. 4(a)* $\epsilon = \frac{1}{5}$*. This minimum value is achieved by (for example) picking the set of 5 vertices* $S = \{a, b, c, d, e\}$*, which is only connected to the rest of the graph by the single edge* (e, f)*. However, in Fig. 4(b) we find that* $\epsilon = \frac{7}{5}$*; any set of 5 vertices is connected by at least 7 edges to the rest of the graph.*

For WSN applications, the network represented by Fig. 4(a) is less desirable, because

- *it is more vulnerable to a listening adversary, who could decrypt a high proportion of communications through the network by the compromise of a single node* e *or* f
- *nodes* e *and* f *are more vulnerable to battery failure*
- *battery failure of just one of the two nodes* e *and* f *would disconnect the network*

– *communication bottlenecks are likely to occur around nodes e and f, making communication through the network less efficient.*

Conversely, in Fig. 4(b) the communication burdens are distributed evenly across the nodes so that battery power will be used more evenly and there are no weak spots for an adversary to target in order to quickly damage the rest of the network. The graph can only be split into disjoint sets by the removal of 4 or more nodes, that is, almost half of the network. It is clear then that Fig. 4(b) represents the less vulnerable WSN.

From this example we see that some strengths and weaknesses of the 'layout' of the WSN are hidden if we only consider the size, key storage, connectivity and resilience, and in Sect. 6 we discuss the practicality of using expansion as another metric for assessing networks. Before that, we consider how best to probabilistically maximise the expansion in an intersection graph, and in Sect. 5 we will consider schemes which aim to produce key graphs with good expansion.

We now consider how to achieve high expansion in an intersection graph $G \cap H$.

Proposition 3. *An intersection graph* $G \cap H = (V \cap V, E_G \cap E_H)$ *has expansion parameter*

$$\epsilon_{G\cap H} \leq \min\{\epsilon_G, \epsilon_H\} \ .$$

Proof. We begin by considering the degree of a node in the intersection graph, which is

$$d_{G\cap H}(v) \leq \min\left(d_G(v), d_H(v)\right)$$

because each edge (v, w) incident to a vertex v in G will be removed in the intersection unless there is also an edge (v, w) in H. Using Prop. 1, we have that $\epsilon_{G\cap H} \leq \min_{v\in V}\{d_{G\cap H}(v)\}$.

Without loss of generality, suppose that $\epsilon_G \leq \epsilon_H$. Consider a set S of vertices in G which achieves the minimum $\frac{|E(S,\overline{S})|}{|S|} = \epsilon_G$. If every edge of $E(S, \overline{S})$ remains in the intersection then $\epsilon_{G\cap H} \leq \epsilon_G$, otherwise $\epsilon_{G\cap H} < \epsilon_G$, since no edges are added elsewhere in the intersection. Therefore we have that $\epsilon_{G\cap H} \leq \min\{\epsilon_G, \epsilon_H\}$. □

We see that it is necessary that G and H have high expansion coefficients for $G \cap H$ to be a good expander. If the communication graph is complete then the expansion of the key graph will be preserved in the intersection. If information about the locations of the nodes is known a priori or if there is some control over the communication graph, then keys can be assigned to nodes in a more efficient manner; see [18] for a survey of KPSs for such scenarios.

However, we usually assume that there is little or no control over the communication graph and model it as a random graph, typically using either the Erdös Rényi model [11] or the random geometric model, as in [5]. If the communication graph is random, all that can be done to aid good expansion in the intersection graph is to design the KPS so that the key graph has as high expansion as possible for a particular network size and for given levels of key storage, connectivity and resilience.

5 Analysing the Expansion Properties of Existing KPSs

Many KPSs produce key graphs with high expansion coefficients for chosen levels of key storage and resilience, as demonstrated by the following examples.

- Random graphs are good expanders with high probability [14], and so Eschenauer and Gligor's random KPS [12] is likely to produce key graphs which are good expanders, as are other random KPSs such as those given in [6].
- Deterministic schemes based on combinatorial designs, as unified in [19], typically guarantee properties such as constant node degree and μ common intersection. That is, if two nodes are not adjacent then they have μ common neighbours, meaning that the graph has diameter 2, and is therefore a good expander. In particular, two deterministic KPSs based on constructions for 'strongly regular' graphs are given in [17].
- Camtepe et al. [5] and Shafiei et al. [21] propose KPSs based on expander graph constructions and demonstrate that these schemes compare well to other well-regarded KPS approaches.

We now consider the KPSs based on expander graph constructions in more detail, and compare them to the other schemes listed above. Camtepe et al. [5] use the Ramanujan construction which produces an 'asymptotically optimal' expander graph (see [14]) for network size $n = t+1$ and key storage $k = s+1$, where t and s are primes congruent to 1 mod 4. Shafiei et al. [21] use the zig-zag construction, which has the benefit of being more flexible to produce key graphs for any sizes of n and k. Both papers use the following method:

1. construct an expander graph G for the appropriate network size and degree
   - in the case of [5], remove any self-loops or multiple edges and replace with randomly-selected edges such that all nodes have the same degree
2. assign a unique pairwise key to every edge of G
3. preload each node with the set of keys which correspond to its set of edges

This ensures that the key graph has high expansion for the chosen network size and node degree r which equals the key storage k.

However, we claim that it is possible to achieve higher expansion in a KPS for the same network size and key storage. This is because in the KPSs based on expander graph constructions, the node degree r is the same as the key storage k, because unique pairwise keys are used. In other KPSs we usually expect that $k < d(v)$ for all vertices v, as illustrated by the following example.

Example 3. In the Eschenauer Gligor random KPS [12], the key storage k is almost certainly less than the degree $d(v)$ of each node $v \in V$ in the key graph. For example, if nodes store 50 keys randomly selected from a pool of 1000 keys, then the expected degree of any node is

$$(n-1) \times \left(1 - \frac{\binom{950}{50}}{\binom{1000}{50}}\right) \quad .$$

If the network has 1000 nodes, this means that the expected degree is ≈ 71.905. *This implies that for the same values of* k *and* n, Pr_1 *is greater in the Eschenauer Gligor scheme than in KPSs produced by expander graph constructions. Since random graphs are known to be good expanders with high probability, this means that, contrary to intuition, a key graph based on an expander graph construction is likely to be a worse expander than a key graph generated by the Eschenauer Gligor scheme.*

Similarly, most schemes based on combinatorial designs also reuse keys so that $k < d(v)$, and therefore produce key graphs with higher average degree and better expansion than those based on expander graph constructions. A benefit of the KPSs based on expander graph constructions is that each key is only used for one edge, meaning that the graphs have perfect resilience: $\mathsf{fail}_s = 0$ for all $1 \leq s \leq n-2$. Therefore, in comparison to many other comparable schemes with perfect resilience, these constructions do produce key graphs with good expansion. However, for fixed values of k and n, if it is desirable to achieve high expansion and higher connectivity at the cost of slightly lowered resilience, then a KPS based on an expander graph construction is not the best choice.

6 Using Expansion as a Metric

We have seen that for two networks of the same size with fixed values of key storage, connectivity and resilience, the WSN represented by the intersection graph with the highest expansion coefficient ϵ is the more robust, with the more evenly distributed flow of data. Therefore we suggest that expansion is an important metric to be considered alongside those listed above, when designing KPSs and assessing their suitability for use in WSNs. However, we now state some drawbacks to the use of expansion as a metric, and explain the extent to which they can be overcome.

Difficulty of determining the expansion coefficient. Determining the expansion coefficient of a given graph is known to be co-NP-complete [3], and so testing KPSs for their expansion coefficient is not an easy task. Additionally, even if the expansion coefficient of the key graph is known, the expansion of the intersection graph will not be known a priori if the communication graph is modelled as a random graph.

Nevertheless, a method for *estimating* the expansion coefficient using the eigenvalues of the incidence matrix of the graph is given in [9], which could be used in the comparison of KPSs. Indeed, if it is possible to determine the locations of the nodes after deployment, for example using an online base station or GPS, it may be feasible to construct the intersection graph and therefore estimate its expansion coefficient, once the WSN has been deployed. This is likely to be relevant if post-deployment key management protocols are available such as key refreshing [2] or key redistribution [10], for which it could be useful to know as much as possible about the vulnerability of the WSN. Some key management protocols are able to provide targeted improvements to specific weak areas of the network, and we explain below how best to identify such weaknesses.

Limitations of the expansion coefficient. We note that the expansion coefficient alone does not claim to fully describe the structure of the graph, giving only a 'worst case' assessment. That is, the value of ϵ only reflects the weakest point of the graph and tells us nothing about the structure of the graph elsewhere.

For example, consider an intersection graph on n nodes which is effectively partitioned into two sets: a set of $n-1$ nodes with high expansion, and a final node which is disconnected from the rest of the graph, as demonstrated in Fig. 5(b). We would find that $\epsilon = 0$, and we would suspect that the graph is less than desirable for WSN applications.

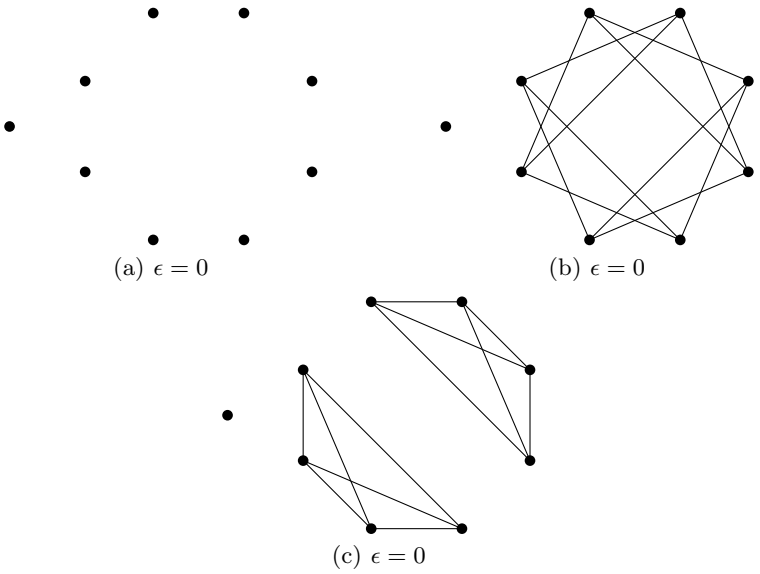


Fig. 5. Distinguishing between cases where $\epsilon = 0$

However, particularly in a network of thousands of nodes, the disconnection of one is unlikely to be severely detrimental to the network; indeed, loss of some nodes due to poor positioning or battery failure may be expected. Knowing only that $\epsilon = 0$ does not distinguish between the following cases:

1. the graph is completely disconnected (Fig. 5(a))
2. a single node is disconnected from the rest of the graph, which otherwise has good expansion (Fig. 5(b))
3. the disconnected graph is a union of smaller graphs, some with good expansion (Fig. 5(c))

If an intersection graph falls into Case 1 then it is likely that the key graph has low connectivity, ie. a low value of Pr_1. However, for the same values of network size, key storage, connectivity and resilience, knowing only that $\epsilon = 0$ in the

intersection graph cannot distinguish between the Cases 2 and 3, though Case 2 is likely to be much better for WSN applications.

Therefore, we suggest some graph-theoretic tools which also serve as indicators of whether the structure of a graph is suitable for WSNs. These may be used alone or in conjunction with (an estimate of) the expansion coefficient in order to analyse a proposed KPS, and where possible to analyse the resulting intersection graph.

Components. We note that to distinguish between the cases in Fig. 5 it is relevant to know the number of *components*. A *component* of a graph is a connected subgraph containing the *maximal* number of edges [8], that is, a subset S of one or more vertices of the graph, where the vertices of S are connected but $E(S, \overline{S}) = \emptyset$. Hence Fig. 5(a) has nine components, Fig. 5(b) has two, and Fig. 5(c) has three. For WSN applications where data must be routed throughout the network, it is desirable to minimise the number of components.

Unlike finding the expansion coefficient of a graph, calculating the number of components can be done in linear time using depth-first search, as described in [15]. The *global connectivity* of a graph is the number of nodes in its largest component divided by the total number of nodes. We wish the global connectivity to be as close to one as possible.

Cut-edges. A *cut-edge* (also known as a *bridge*) is an edge whose deletion increases the number of components. Equivalently, an edge is a cut-edge if it is not contained in any cycle of the graph. This is illustrated in Fig. 4, where the edge (e, f) is a cut-edge.

As we have seen, cut-edges in the intersection graph for a WSN are undesirable because they can cause bottlenecks, increase communication burdens on the nodes at their endpoints, and create weak points of the network where a small fault or compromise by an adversary creates a lot of damage. Therefore, one of the reasons why intersection graphs with high expansion are desirable for WSNs is because they are less likely to have cut-edges:

- If $\epsilon > \frac{1}{\lfloor \frac{|V|}{2} \rfloor}$ then we know that there is no cut-edge which, if removed, would separate the graph into two components, each of size $\frac{|V|}{2}$.
- If $\epsilon = \frac{1}{2}$ then it is possible that there are cut-edges which, if removed, would disconnect at most two nodes from the network
- If $\epsilon > 1$ then for all $S \subset V$ with $|S| \leq \frac{|V|}{2}$,

$$|E(S, \overline{S})| > |S| \geq 1 \ ,$$

and so there can be no cut-edges in the graph.

Determining whether a graph contains cut-edges can also be done by a linear time algorithm [22].

Cutpoints. There is also a related notion of *cutpoints* in graphs, for which we will need the following definitions from [8]. A *subgraph* of a graph $G(V, E)$ is a graph $G_S(V_S, E_S)$ in which $V_S \subset V$ and $E_S \subset E$. If V_S or E_S is a proper subset (that is, $V_S \neq V$ or $E_S \neq E$), then the subgraph is a *proper subgraph* of G. If V_S or E_S is empty, the subgraph is called the *null graph*.

In a connected graph G, if there exists a proper non-null subgraph G_S such that G_S and its complement have only one node n_i in common, then the node n_i is called a *cutpoint* of G. In an unconnected graph, a node is called a cutpoint if it is a cutpoint of one of its components. If G has no self-loops, then a cutpoint is a node whose removal increases the number of components by at least one. We see then that in Fig. 4(a), nodes e and f are cutpoints, and that graphs with good expansion will have few cutpoints.

If a graph contains no cutpoints it is said to be *nonseparable* or *biconnected*, which again is clearly desirable for an intersection graph representing a WSN. The website [1] gives examples of Java algorithms which find the nonseparable components of given graphs and can even add edges to make graphs nonseparable.

These tools are just a few of the simple, effective ways to analyse an intersection graph of a deployed WSN, and to make intelligent improvements to the structure of the graph wherever the post-deployment key management protocols allow.

7 Conclusion

We have shown that if we fix levels of key storage, network size, connectivity and resilience, then the larger the value of the expansion coefficient ϵ in the intersection graph, the better suited it will be for WSNs. This is because graphs with good expansion are well connected with low diameter and do not have the vulnerabilities of cut-edges and cutpoints. We have shown that the expansion coefficient of the product graph is not a relevant metric, but that it is the intersection graph where good expansion is desired.

In a setting where there is control over the communication graph, the expansion of the intersection graph should be an important consideration in the design of the key graph. If there is no control over the communication graph, then after choosing levels of network size, key storage, connectivity and resilience, the best choice of KPS is the one with the highest expansion, since it will maximise the probability of achieving good expansion in the intersection graph.

We have shown that KPSs based on expander graph constructions are able to produce key graphs with high expansion for a given network size and key storage, and use unique pairwise keys to give perfect resilience. However, many existing KPSs are able to achieve better expansion for the same key storage and network size, at the cost of lower resilience.

Finally, we have suggested that expansion is an important metric for comparing KPSs proposed for WSNs, and a useful parameter for analysing intersection graphs after deployment in order to improve weak parts of the network. Determining the expansion of a graph is co-NP-complete and gives only a worst-case

assessment of the graph. Therefore we have proposed the use of linear time algorithms to estimate the expansion, and introduced related graph-theoretic properties which could be used to analyse the key and intersection graphs of WSNs.

References

1. Analyzing Graphs, http://docs.yworks.com/yfiles/doc/developers-guide/analysis.html
2. Blackburn, S.R., Martin, K.M., Paterson, M.B., Stinson, D.R.: Key Refreshing in Wireless Sensor Networks. In: Safavi-Naini, R. (ed.) ICITS 2008. LNCS, vol. 5155, pp. 156–170. Springer, Heidelberg (2008)
3. Blum, M., Karp, R.M., Vornberger, O., Papadimitriu, C.H., Yannakakis, M.: The Complexity of Testing whether a Graph is a Superconcentrator. Information Processing Letters 13(4-5), 164–167 (1981)
4. Camtepe, S.A., Yener, B.: Key Distribution Mechanisms for Wireless Sensor Networks: a Survey. Rensselaer Polytechnic Institute, Computer Science Department, Tech. Rep. TR-05-07 (2005)
5. Camtepe, S.A., Yener, B., Yung, M.: Expander Graph Based Key Distribution Mechanisms in Wireless Sensor Networks. In: ICC 2006, IEEE International Conference on Communications, pp. 2262–2267 (2006)
6. Chan, H., Perrig, A., Song, D.: Random Key Predistribution Schemes for Sensor Networks. In: SP 2003: Proceedings of the 2003 IEEE Symposium on Security and Privacy, pp. 197–213. IEEE Computer Society, Washington, DC (2003)
7. Chen, C.-Y., Chao, H.-C.: A Survey of Key Distribution in Wireless Sensor Networks. In: Security and Communication Networks (2011)
8. Chen, W.-K.: Applied Graph Theory. University of Virginia, North-Holland (1971)
9. Chung, F.R.K.: Spectral Graph Theory. American Mathematical Society, California State University, Fresno (1994)
10. Cichoń, J., Gołębiewski, Z., Kutyłowski, M.: From Key Predistribution to Key Redistribution. In: Scheideler, C. (ed.) ALGOSENSORS 2010. LNCS, vol. 6451, pp. 92–104. Springer, Heidelberg (2010)
11. Erdös, P., Rényi, A.: On the Evolution of Random Graphs. Publication of the Mathematical Institute of the Hungarian Academy of Sciences, 17–61 (1960)
12. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: CCS 2002: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 41–47. ACM, New York (2002)
13. Ghosh, S.K.: On Optimality of Key Pre-distribution Schemes for Distributed Sensor Networks. In: Buttyán, L., Gligor, V.D., Westhoff, D. (eds.) ESAS 2006. LNCS, vol. 4357, pp. 121–135. Springer, Heidelberg (2006)
14. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin of the American Mathematical Society 43(04), 439–562 (2006)
15. Hopcroft, J., Tarjan, R.: Algorithm 447: efficient algorithms for graph manipulation. Commun. ACM 16(6), 372–378 (1973)
16. Kleinberg, J., Rubinfeld, R.: Short Paths in Expander Graphs. In: Annual Symposium on Foundations of Computer Science, vol. 37, pp. 86–95 (1996)
17. Lee, J., Stinson, D.R.: Deterministic Key Predistribution Schemes for Distributed Sensor Networks. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 294–307. Springer, Heidelberg (2004)

18. Martin, K.M., Paterson, M.B.: An Application-Oriented Framework for Wireless Sensor Network Key Establishment. Electronic Notes in Theoretical Computer Science 192(2), 31–41 (2008)
19. Paterson, M.B., Stinson, D.R.: A unified approach to combinatorial key predistribution schemes for sensor networks. Cryptology ePrint Archive, Report 076 (2011)
20. Reingold, O., Vadhan, S., Wigderson, A.: Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, pp. 3–28. IEEE Computer Society, Washington, DC (2000)
21. Shafiei, H., Mehdizadeh, A., Khonsari, A., Ould-Khaoua, M.: A Combinatorial Approach for Key-Distribution in Wireless Sensor Networks. In: Global Telecommunications Conference, IEEE GLOBECOM 2008, pp. 1–5. IEEE (2008)
22. Tarjan, R.: A Note on Finding the Bridges of a Graph. Information Processing Letters, 160–161 (1974)
23. Xiao, Y., Rayi, V.K., Sun, B., Du, X., Hu, F., Galloway, M.: A survey of key management schemes in wireless sensor networks. Computer Communications 30(11-12), 2314–2341 (2007)

Γ-MAC[H, P] - A New Universal MAC Scheme

Ewan Fleischmann, Christian Forler, and Stefan Lucks

Chair of Media-Security, Bauhaus-University Weimar, Germany
{Ewan.Fleischmann,Christian.Forler,Stefan.Lucks}@uni-weimar.de

Abstract. In this paper, we introduce a new class of universal hash function families called almost regular universal (ϵ-ARU). Informally, an ϵ-ARU hash function family is almost universal, and additionally provides almost regularity. Furthermore, we present Γ-MAC[H, P], a new MAC scheme based on a ϵ-ARU hash function family. It is the first stateless MAC scheme based on universal hash functions, which requires only one n-bit key. Γ-MAC[H, P] is provable secure and an alternative to the Wegman-Carter-Shoup (WCS) based MAC scheme, where the security breaks apart in the nonce-reuse scenario [11,28].

In addition, we show that Γ-MAC[H, P] can be implemented very efficiently in software. For messages longer than one kilobyte, our Γ-MAC[H, P] implementation is even faster than the optimized AES-128 implementations from Schwabe and Bernstein from the eBash project[1].

Keywords: universal hashing, provable security, message authentication code.

1 Introduction

Message Authentication Code. A message authentication code (MAC) is a widely used cryptographic primitive – utilized to verify the integrity and authenticity of messages, assuming a secret key k shared by sender and receiver. A MAC scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ consists of three algorithms $\mathcal{K}$, $\mathcal{T}$, and $\mathcal{V}$. The randomized key generation function $\mathcal{K}$ takes no input and returns a random key K. The authentication function $\mathcal{T}$ is used by the sender to generate a security tag $\tau = \mathcal{T}_k(m)$ for a message m. Given the pair (m, τ), the receiver calls the verification function $\mathcal{V}_k(m, t)$, which returns `true` if τ is the corresponding security tag of m. We require that $\mathcal{V}_K(M, \mathcal{T}_K(M)) = \texttt{true}$ holds for any given parameters.

The security tag τ ought to be short (typically 32–256 bit) to minimize the overhead for authentication. An adversary attempts to forge a message, i.e., to find new tuples (m', τ') with $\mathcal{V}_k(m', \tau') = \texttt{true}$. We consider chosen-plaintext existential forgery attacks, where the adversary is allowed to freely choose messages (and nonces), and succeeds by forging the tag for any new message. A MAC is called secure, if it is hard for the adversary to succeed.

[1] http://cr.yp.to/streamciphers/timings/estreambench-20080905.tar.bz2

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 83–98, 2012.

Universal hashing. Information-theoretically secure MACs have first been studied by Gilbert, MacWilliams and Sloane [26], and later by Wegman and Carter [45]. A strictly information-theoretical approach would require very long keys or greatly limit the number of messages to be authenticated under a given key. Thus, one typically combines the information-theoretical part – a universal hash function – with an additional function, which is modelled as a random function or permutation. There are two families of such MACs, which were studied so far.

Universal MAC schemes. The first family is due to Wegman, Carter and Shoup. We denote it as the "Wegman-Carter-Shoup" [43] (in short: "WCS$[H, F]$") approach. The WCS$[H, F]$-MAC is based on a family of ϵ almost XOR universal hash functions (ϵ-AXU) H, and a family of pseudorandom functions (PRF) F. For $h \in H$ and $f \in F$ the WCS-MAC is defined as $\mathrm{WCS}_{f,h}(m, z) = h(m) \oplus f(z)$, where m is the message and z is the nonce. The WCS approach allows to use an ϵ-AXU hash function h several times, if and only if the nonce is never reused. The nonce is essential for the security of this MAC scheme. Reusing it leads to serious security breaches [28]. MACs following the WCS approach are well studied and improved over the years for cryptographic purposes by Bernstein [4,5] Brassard [18], Krawczyk [35], Rogaway [41], Stinson [44], and other authors [5,12,16,24,27,38]. The second family follows the UMAC$[H, F]$ resp. FCH paradigm (we read this as "Function, Concatenation, Hash") from Black et al. [12]. We call MACs that fit into this scheme WMAC$[H, F]$, like Black and Cochran in [11]. Let H be a family of ϵ almost universal (ϵ-AU) hash functions and F a family of PRFs. For $h \in H$ and $f \in F$ the WMAC is defined as $\mathrm{WMAC}_{f,h}(m, z) := f(h(m), z)$ for a message m and a nonce z. Alike, as WCS$[H, F]$, the message is hashed using a randomly chosen hash function h out of a family of universal hash functions. In contrast to WCS, the hash output is not XOR-ed with the output of a random function, but is used as a part of the random function's input, jointly with the nonce z. Since the internal hash values $h(m)$ are only used as the input for a random function, the security of WMAC$[H, F]$ remains intact even if the nonce z is re-used. In fact, one doesn't actually need a nonce, but can securely use $\tau = f(h(m))$. Moreover, most messages have meta data such as a timestamp or a sequence counter. This auxiliary information can be used as "nonce" to guarantee defense against "replay attacks" – i.e., resending an old message and its corresponding authentication tag. Here, the adversary hopes that the replayed message will be misunderstood in the new context.

Our Contribution. At first we introduce ϵ-APU, a new class of universal hash functions. Informally we say a family of hash functions H is ϵ-APU, for a randomly chosen $h \in H$, if 1) the output of h is almost uniformly distributed, 2) it is very unlikely that to distinct inputs m and m' collide ($h(m) = h(m')$). Then we present Γ-MAC$[H, P]$, a stateless MAC scheme based on a family of ϵ-APU hash functions H and a pseudorandom permutation p. A family of permutations $P : \mathcal{K}\{0,1\}^n \to \{0,1\}^n$ is called PRP, if $p(k, .)$ is "computationally indistinguishable" from a random permutation on $\{0,1\}^n$. For sake of

Table 1. Overview of published MAC Schemes. The Domain is normalized to $(\{0,1\}^n)^l$. We denote *Invocations* as the sum of all BC, PRNG, and compression function calls as well as all n-bit multiplications. Note, a i/nj-bit multiplication counts an i/nj invocation.

MAC	Invocations	Keylen	Stateless
CBC-MAC [39]	l	n	Yes
CWC-MAC [33]	$l+3$	n	No
DMAC [40]	$l+1$	$2n$	Yes
ECBC [13]	$l+1$	$3n$	Yes
FCBC [13]	l	$3n$	Yes
GMAC [38]	$l+3$	n	No
Γ^*-MAC$[H, P]$	$l+1$	n	Yes
HMAC [2]	$l+3$	n	Yes
OMAC [30]	$l+1$	n	Yes
PMAC [14]	$l+1$	n	Yes
Poly1305-AES [5]	$l+1$	$2n$	No
RMAC [23]	$l+2$	$2n$	No
TMAC [37]	$l+1$	$2n$	Yes
UMAC8-128 [36]	$3l+1$	$2l+1$	No
WC-MAC [45]	$l+1$	$ln+1$	No
WMAC [11,28]	$l+1$	$2n$	No
XCBC [13]	l	$3n$	Yes

convenience, we usually write p_k for $p(k, .)$. For $h \in H$, Γ-MAC$[H, P]$ is defined as $\Gamma_{h(m)} := p_{h(m)}(|m|)$. In contrast to WCS$[H, F]$ and WMAC$[H, F]$, our MAC scheme needs only one key. This key determines unambiguously the universal hash function h from H. The PRP p is determined unambiguously from the output value of h. Γ-MAC$[H, P]$ is the second stateless universal MAC scheme – beside WMAC$[H, F]$– that is known in literature. The major advantage of Γ-MAC$[H, P]$ over WMAC$[H, F]$ is that we only need *one* key. On closer look, this can be also a disadvantage because the used block cipher must be resistant against related key attacks. Therefore, we present a ϵ-AXPU hash families. Instantiated with such a hash family the used block cipher must not guarantee resistance against related key attacks that exploit a XOR difference between keys. Furthermore, the key scheduler is invoked each time if the authentication or verification function is called. Our analysis showed that the performance of Γ_X-AES, a Γ-MAC$[H, P]$ instance based on AES-128, does not suffer significantly from this key scheduler invocation issue.

Related Work. Modern universal hash function based MACs, like GMAC from McGrew and Viega [38], VMAC from Krovetz and Dai [20,36] or Bernstein's Poly1305-AES [5] are similar in spirit to Γ-MAC$[H, P]$, but employ two n-bit keys, while a single one suffices for Γ-MAC$[H, P]$. Like GMAC, our Γ-MAC$[H, P]$ instance uses a universal hashing function familiy based on Galois field multiplications. Handschuh and Preneel [28] pointed out that MACs based on universal hashing are brittle, with respect to their combinatorial

```
1  Initialize()
2    k ← K();

3  Finalize()
4    return win;

10 Tag(M)
11   τ ← T_K(M);
12   S ← S ∪ {(M, τ)};
13   return τ;

20 Verify(M, τ)
21   r ← V_K(M, τ);
22   if (r = true and (M, τ) ∉ S) then
23     win ← true;
24   return r;
```

Fig. 1. Game SUF-CMA$_{\mathcal{MA}}$ where $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$

properties, and that some are extremely vulnerable to nonce reuse attacks. Black and Cochran [11] recently presented WMAC$[H, F]$. This MAC scheme matches the security bounds given by the best known attacks from Handschuh and Preneel.

Table 1 compares the key length of known MAC schemes and the number of basic operations such as block cipher invokation or n-bit multiplications. Furthermore, it indicates whether a MAC is stateless or not.

Organization of the paper. Section 2 introduces the security notions of MAC algorithms, and universal hash functions. Section 3 provides the formal definition of ϵ-ARU, ϵ-AXRU and Γ-MAC$[H, P]$ including security proofs. Section 4 introduces Γ_X-AES a Γ-MAC$[H, P]$ instance, and in Section 5 their security is proven. Section 6 is all about the software performance of an optimized Γ_X-AES implementation. The benchmarking took place on a Intel Core i5 M540 2.53 GHz. Finally, in Section 7 of this paper, we summarize our results and conclude.

2 Preliminaries

Notion. Let $\{0,1\}^n$ denote the set of all n binary string, and $\{0,1\}^*$ the set of all binary strings of arbitrary length. Let $a, b \in bset*$, then $a||b$ denotes the concatenation of a and b. For a set A we denote $|A|$ the order of A, and if $a \in A$, the length of a in bits is denoted $|a| = \lceil \log_2(A) \rceil$. Finally, $\{0,1\}^{n*}$ denotes the set that of all binary strings whose length in bits is a multiple of n, including $\{0,1\}^n$.

2.1 Security Notions

The insecurity of a MAC scheme is quantified by the success probability of an optimal resource-bounded adversary. The resource is the number of queries to the functions `Tag` and `Verify` - from SUF-CMA game described in Figure 1 - hereafter referred as *tag oracle* Q_T and as *verify oracle* Q_V. An adversary is a computationally unbounded but always-halting algorithm A with access to Q_v and Q_t

The security of a MAC scheme $\mathcal{MA}$ =($\mathcal{K}$,$\mathcal{T}$,$\mathcal{V}$) is measured by the success probability in the of winning the strong unforgeability against chosen message attack (SUF-CMA) game described in Figure 1. We denote $\mathbf{Adv}^{\text{SUF-CMA}}_{\mathcal{MA}}(A) = \Pr[A^{\text{SUF-CMA}}_{\mathcal{MA}} \Rightarrow 1]$ as the probability that function `Finalize` returns true, after A has made all its requests.

Informally, $\mathcal{MA}$ is SUF-CMA secure if for every reasonable adversary A, the probability to win the SUF-CMA game is suitably small. In the context of this paper, we call an adversary reasonable, if it invokes Q_T and Q_V at most $2^{n/4}$ times.

We denote $\mathbf{Adv}^{\text{SUF-CMA}}_{\mathcal{MA}}(q_s, q_v, t, \ell)$ as the maximal value of $\mathbf{Adv}^{\text{SUF-CMA}}_{\mathcal{MA}}(A)$ among adversaries that run in time at most t, query at most q_s times the `tag oracle` Q_T, query at most q_v times `verify oracle` Q_V, and the total number of n-bit blocks of all requested messages is at most l.

2.2 Universal Hash Families

Universal hash families (UHF) $H = \{h : \mathcal{M} \to \mathcal{T}\}$ are used frequently in building blocks for message authentication codes [5,11,15,24,27,33,36,38,41,45]. They guarantee a low number of collisions in expectation, even if the data is chosen by an adversary. In the following, we define several notions that are needed later.

Definition 1 ([45] ϵ-almost universal hash functions (ϵ-AU)) *Let $H = \{h : \mathcal{M} \to \mathcal{T}\}$ be a family of hash functions and $\epsilon \in \mathbb{R}$ with $1/|\mathcal{T}| \leq \epsilon \leq 1$. We say H is ϵ-almost universal, written ϵ-AU, if for all distinct message pairs $(m, m') \in \mathcal{M} \times \mathcal{M}$ with $m \neq m'$*

$$\Pr_{h \in H}[h(m) = h(m')] \leq \epsilon.$$

Note, that the probability only holds for input pairs, and ϵ specifies the degree of uniform distribution.

Definition 2 ([35] ϵ-almost XOR universal hash functions (ϵ-AXU)) *Let $H = \{h : \mathcal{M} \to \{0,1\}^n\}$ be a family of hash functions, and $\epsilon \in \{0,1\}^n$ with $1/2^n \leq \epsilon \leq 1$. We say H is ϵ-almost XOR universal, written ϵ-AXU, if for all distinct $m, m' \in \mathcal{M}$, and for all $c \in \{0,1\}^n$*

$$\Pr_{h \in H}[h(m) \oplus h(m') = c] \leq \epsilon.$$

The definition can be simply adapted to any group by exchanging XOR with the specific group operation. In the current paper, we focus on arithmetic in GF(2^n) and thus we only need to consider XOR-differences.

Note, that each ϵ-AXU hash family H is ϵ-AU, because for $c = 0$ we have

$$\Pr_{h \in H}[h(m) \oplus h(m') = 0] = \Pr_{h \in H}[h(m) = h(m')] \leq \epsilon.$$

2.3 Universal MAC Schemes

The WCS-MAC Scheme

Definition 3 ([43,45] WCS$[H, F]$) *Let $H = \{h : \mathcal{M} \to \{0,1\}^r\}$ be a family of ϵ-AXU hash functions and $F = \{f : \mathcal{Z} \to \{0,1\}^r\}$ be a family of pseudo-random functions. Let $h \in_R H$, $f \in_R F$,*

$m \in \mathcal{M}$ a message and $z \in \mathcal{Z}$ be a nonce. The notation $e \in_R S$ implies that S is a finite set and e is chosen (uniformly) at random from S. The Wegman-Carter-Shoup-MAC (WCS-MAC) is defined as

$$WCS\text{-}MAC_{h,f}(m,z) = h(m) \oplus f(z)$$

and

$$VF\text{-}WCS_{h,f}(m,z,\tau) = \begin{cases} \texttt{true} & \tau = h(m) \oplus f(z), \\ \texttt{false} & else. \end{cases}$$

Lemma 1 ([38] WCS$[H,F]$ security)
Let $q = q_v = q_s$ then

$$\mathbf{Adv}^{\text{SUF-CMA}}_{WCS_{[H,F]}}(q,q,\infty,\ell) \leq q\epsilon.$$

This lemma implies that WCS$[H,F]$ is secure – even for adversaries that have unlimited computation power – while it is not possible to determine which hash function $h \in H$ is used. The "encryption" of the hash value $h(m)$ by XOR $f(z)$ makes it impossible to determine which hash function $h \in H$ is used.

The WMAC Scheme

Definition 4 ([12] WMAC$[H,F]$)
Let $F = \{f : A \to \mathcal{T}\}$ be a family of random functions, $H = \{h : \mathcal{M} \to A\}$ be an ϵ-AU family of hash functions, $f \in_R F$ and $h \in_R H$. Let $m \in \mathcal{M}$ a message then we define

$$WMAC_{h,f}(m) = f(h(m))$$

and

$$WMAC\text{-}VF_{h,f}(m,\tau) = \begin{cases} \texttt{true} & \tau = f(h(m)), \\ \texttt{false} & else. \end{cases}.$$

Lemma 2 ([12] WMAC security)

$$\mathbf{Adv}_{WMAC[H,F]}(q_s,q_v,\infty,\ell) \leq q_v(q_s^2\epsilon + |\mathcal{T}|).$$

3 The Gamma-MAC Scheme

In this section, we present two specifications of the Gamma-MAC schemes: Γ-MAC$[H,P]$ and Γ_X-MAC$[H,P]$. We use the term Gamma-MAC as the generic name for both schemes.

Definition 5 (ϵ-almost regular universal hash functions (ϵ-ARU))
Let $H = \{h : \mathcal{M} \to \mathcal{T}\}$ be an ϵ-AU hash function family. We say H is ϵ-almost regular universal written ϵ-ARU, if for all $m \in \mathcal{M}$ and $c \in \mathcal{T}$

$$\Pr_{h \in H}[h(m) = c] \leq \epsilon.$$

Definition 6 (Γ-MAC$[H, P]$)
Let $H = \{h : \mathcal{M} \to \mathcal{K}\}$ be an ϵ-ARU family of hash functions, $P = \{g : \mathcal{K} \times \mathcal{T} \to \mathcal{T}\}$ be a family of pseudorandom permutations, $m \in \mathcal{M}$ and $|m|$ the bit length of m. Then we define

$$\Gamma\text{-}MAC_{h,P}(m) = p_{h(m)}(|m|)$$

and

$$\Gamma\text{-}VF_{h,P}(m, \tau) = \begin{cases} \texttt{true} & \tau = p_{h(m)}(|m|), \\ \texttt{false} & else. \end{cases}.$$

Theorem 1 (Γ-MAC$[H, P]$ security)
Let $H = \{h(m) : \{0,1\}^{n^*} \to \mathcal{K}\}$ be an ϵ-ARU hash function family and $P = \{p_k : \mathcal{K} \times \mathcal{T} \to \mathcal{T}\}$ a family of pseudo-random permutations. Then,

$$\mathbf{Adv}^{\text{SUF-CMA}}_{\Gamma\text{-}MAC[H,P]}(q_s, q_v, t, \ell) \leq q_v \left(q_s\epsilon + q_s(q_s - 1)\epsilon + \frac{2 + q_s(q_s - 1)}{2|\mathcal{T}|} \right).$$

Note, that $m \in \{0,1\}^{n^*}$ is equivalent with n divides $|m|$.

Proof
In this proof we make a case analysis to upper bound the success probability for adversaries with $q_v = 1$. The generalization to $q_v \geq 1$ verification queries increase the success probability for any adversary at most by q_v. So, we have

$$\mathbf{Adv}^{\text{SUF-CMA}}_{\Gamma\text{-MAC}[H,P]}(q_s, q_v, t, \ell) \leq q_v \cdot \mathbf{Adv}^{\text{SUF-CMA}}_{\Gamma\text{-MAC}[H,P]}(q_s, 1, t, \ell).$$

Case 1: The adversary wins if it is able two find two distinct authentication queries m_i and m_j $(1 \leq i < j \leq q_s)$ that are maped to the same authentication tag. This means that $\tau_i = p_{x_i}(|m_i|) = p_{x_j}(|m_j|) = \tau_j$ for $x_i = h(m_i)$ and $x_j = h(m_j)$. Therefore,

$$\begin{aligned} &\mathbf{Pr}_{h\in H}[\tau_i = \tau_j] \\ &= \underbrace{\mathbf{Pr}_{h\in H}[\tau_i = \tau_j \mid x_i = x_j] \cdot \mathbf{Pr}[x_i = x_j]}_{E_1} + \underbrace{\mathbf{Pr}_{h\in H}[\tau_i = \tau_j \mid x_i \neq x_j] \cdot \mathbf{Pr}[x_i \neq x_j]}_{E_2}. \end{aligned}$$

Subcase E_1: We distinguish between $|m_i| = |m_j|$ and $|m_i| \neq |m_j|$.

- If $|m_i| = |m_j|$ then for two distinct messages m_i and m_j is

  $$\mathbf{Pr}_{h\in H}[\tau_i = \tau_j \mid x_i = x_j] = \mathbf{Pr}_{h\in H}[p_{x_i}(|m_i|) = p_{x_j}(|m_j|) \mid x_i = x_j] = 1.$$

  The probability for the event "$x_i = x_j$" is equivalent to

  $$\mathbf{Pr}_{h\in H}[h(m_i) = h(m_j)] \leq \frac{q_s(q_s - 1)\epsilon}{2} \qquad 1 \leq i < j \leq q_s.$$

- Else if $|m_i| \neq |m_j|$ the probability for this case is zero, since $p_{x_i}(|m_i|) \neq p_{x_j}(|m_j|)$ for all $|m_i|, |m_j|, x_i, x_j \in \{0,1\}^n$ and $p \in P$ for $|m_i| \neq |m_j|$ and $x_i = x_j$.

Subcase E_2: Let $x_i, x_j \in \{0,1\}^n$ be fixed and randomly chosen. In this case we assume that $\mathbf{Pr}[x_i \neq x_j] \leq 1$. For any tuple $(z_i, z_j) \in \mathcal{T} \times \mathcal{T}$ we have

$$\mathbf{Pr}[p_{x_i}(z_i) = p_{x_j}(z_j)] = 1/|\mathcal{T}|.$$

Then, for $1 \leq i < j \leq q_s$ we have

$$\mathbf{Pr}[p_{x_i}(|m_i|) = p_{x_j}(|m_j|)] \leq \frac{q_s(q_s-1)}{2|\mathcal{T}|}.$$

Case 2: The adversary wins if it is able two find two distinct authentication queries m_i and m_j $(1 \leq i < j \leq q_s)$ that collide under h. Since H is ϵ-ARU, we can upper bound the success probability for this case by $\frac{q_s(q_s-1)}{2|\mathcal{T}|}$.

Case 4: The success probability for any adversary to guess the right output $h(m)$ for any verification query (m, τ) is at most ϵq_s, since H is ϵ-ARU.

Assume that an adversary A does not win due to one of this three cases. this means that A is now facing the task to guess the output of p under a new key k. The success probability therefore is at most $1/|\mathcal{T}|$.

Our claim follows by adding up the individual bounds. □

Let consider the scenario where $= q_s = q_v$ and $\epsilon > |\mathcal{T}|$. Here, the term $\frac{q(q-1)\epsilon}{2}$ determines the security of Gamma-MAC and we have $\mathbf{Adv}^{\text{SUF-CMA}}_{\Gamma\text{-MAC}_{[H,P]}}(q, q, t, \ell) \leq O(q^3\epsilon)$, which corresponds approximately to the security of a stateless WMAC based MAC scheme. The security bound for the statefull WCS-MAC scheme $O(q\epsilon)$ is significant better. This security is achieved by using a nonce. Though the concept of a nonce is simple in theory, it is a challenge to ensure that a nonce is only used once, *ever*. Flawed implementations of nonces are ubiquitous [17, 29, 32, 42, 46]. Apart from implementation failures, there are fundamental reasons why application programmers can't always prevent nonce reuse. Consider a counter or a random generator state stored persistently and reloaded after a restart. How should the application programmer defend old data from being restored during a backup, or current date being duplicated if a virtual machine is cloned? The point of a virtual machine that it should hide its virtualness from the applications.

Therefore we advise against the use of a statefull MAC scheme.

Besides the collision resistance, ϵ-ARU hash function families do not guarantee any other properties about the output relation of two distinct inputs. Theoretically, this does not matter, since Bellare and Cash showed that a PRP is provable secure against related-key attacks [3], where the adversary can partially control an unknown key. But, in practice, a PRP is mostly implemented with a block cipher which can be vulnerable to such an attack, due to weaknesses in the key schedule. In recent years it has been shown that common used block ciphers like AES [19] or KASUMI [1] are vulnerable to related-key attacks [8–10, 22]. The majority of related-key attacks consider XOR differences between keys [8–10, 21, 22, 25, 31, 34], what can be interpreted as flipping of certain key bits.

Hence, the presented Γ-MAC$[H, P]$ scheme does not provide protection against such attacks. For this reason, we present Γ_X-MAC$[H, P]$, a Gamma-MAC scheme that offers protection against related-key attacks based on XOR differences.

Definition 7 (ϵ-almost XOR regular universal hash functions (ϵ-AXRU))
Let $H = \{h : \mathcal{M} \to \mathcal{T}\}$ be an ϵ-AXU hash function family. We say H is ϵ-almost XOR regular universal written ϵ-AXRU, if for all $m \in \mathcal{M}$ and $c \in \mathcal{T}$

$$\Pr_{h \in H}[h(m) = c] \leq \epsilon.$$

Definition 8 (Γ_X-MAC$[H, P]$)
Let $H = \{h : \mathcal{M} \to \mathcal{K}\}$ be an ϵ-AXRU family of hash functions, $P = \{g : A \times \mathcal{K} \to B\}$ be a family of keyed random permutations, and $m \in \mathcal{M}$. Then we define

$$\Gamma_X\text{-}MAC_{h,P}(m) = p_{h(m)}(|m|)$$

and

$$\Gamma_X\text{-}VF_{h,P}(m, \tau) = \begin{cases} \texttt{true} & \tau = p_{h(m)}(|m|), \\ \texttt{false} & else. \end{cases}$$

Theorem 2 (Γ_X-MAC$[H, P]$ security)
Let $H = \{h(m) : \{0,1\}^{n^*} \to \mathcal{K}\}$ be an ϵ-AXRU hash function family and $P = \{p_k : \mathcal{K} \times \mathcal{T} \to \mathcal{T}\}$ a family of pseudo-random permutations. Then,

$$\mathbf{Adv}^{\text{SUF-CMA}}_{\Gamma_X\text{-}MAC[H,P]}(q_s, q_v, t, \ell) \leq q_v \left(q_s\epsilon + q_s(q_s - 1)\epsilon + \frac{2 + q_s(q_s - 1)}{2|\mathcal{T}|} \right).$$

The proof is similar to the proof of Theorem 1.

4 Gamma-MAC Instance: Γ_X-AES

The Γ_X-AES MAC is based on the common block chaining approach from the CBC-MAC [13]. It takes as input a message of arbitrary length and produces as output a 128-bit security tag τ. The message $m = m_1, ..., m_b$ is divided into b blocks with $|m_i| = 127$ for $i = 1, ..., b - 1$, and $|m_b| \leq 127$. If necessary, the final message block m_b is extended to 127 bits by appending the bit sequence $10^{126-|m_b|}$. A chaining value c_i is recursively defined by $c_i = h(m_i \oplus c_{i-1})$. This allows to use a single hash function $h : \{0,1\}^n \to \{0,1\}^n$ to authenticate messages of arbitrary length.

Definition 9 (Γ_X-AES)
Let $G_{127} = \{g_k : GF(2^{127}) \setminus \{0,1\} \times GF(2^{127}) \to GF(2^{127})\}$ with $g_k(x) = k \cdot x$ for $k \in GF(2^{127}) \setminus \{0,1\}$ and $x \in GF(2^{127})$. Let $m = m_1, ..., m_b$ with $|m_i| = 127$

for $i = 1, ..., b$, *and* $G = \{h_k : GF(2^{127}) \setminus \{0,1\} \times GF(2^{127})^b \to GF(2^{127})\}$ *a hash function family defined by*

$$c_0 := g_k(1), \quad c_i := g_k(m_i \oplus c_{i-1}) \qquad i = 1, ..., b$$

$$h_k(m) := c_b = k^{b+1} \sum_{i=1}^{b} k^{b+1-i} m_i.$$

Let m *be a message of arbitrary length and* $|m|$ *a 128-bit value that contains the message length of* m *in bits. Then* Γ_X*-AES is defined as follows:*

$$\Gamma_X\text{-}MAC_{G,\mathrm{AES}}(m) := \mathrm{AES}_{G(m)||0}(|m|)$$

and

$$\Gamma_X\text{-}VF_{G,\mathrm{AES}}(m,\tau) = \begin{cases} \texttt{true} & \tau = \Gamma_X\text{-}MAC_{G,\mathrm{AES}}(m), \\ \texttt{false} & \textit{else.} \end{cases}$$

5 Security Analysis of Γ_X-AES

In this section we prove that Γ_X-AES is SUF-CMA secure. For this it suffices to show that the hash function family $G : \mathrm{GF}(2^{127}) \setminus \{0,1\} \times \mathrm{GF}(2^{127})^* \to \mathrm{GF}(2^{127})$ as defined in Definition 9 is ϵ-AXRU. Then we can apply Theorem 1 which tells us that Γ_X-AES is SUF-CMA secure.

Theorem 3 (G is ϵ-AXRU)
The hash function family G *from Definition 9 is for* $b = 1, ..., 2^{127} - 2$ ϵ*-AXRU with* $\epsilon = \frac{b+1}{2^{127}-2}$.

Proof. From Lemma 3 and 4 follows that G is ϵ-AXU.
Let $m \in \mathrm{GF}(2^{127})^b$ with $m = m_1, ..., m_b$, and $c \in \mathrm{GF}(2^{127})$. Beginning with $G(m) = c$, and moving c on the right side of the equation we get $\underbrace{k^{b+1} \sum_{i=1}^{b} k^{b+1-i} m_i}_{G(m)} - c = 0$. The none-zero polynominal – on the left side of this equation – has at most $b+1$ roots. Therefore, we have $\Pr_{k \in \mathrm{GF}(2^{127}) \setminus \{0,1\}}[G(m) = c] \leq \frac{b+1}{2^{127}-2}$. Note, the observation above holds for any tuple $(m, c) \in \mathrm{GF}(2^{127})^b \times \mathrm{GF}(2^{127})$. □

Lemma 3 (Galois field multiplication)
Let $G_{127} = \{g_k : GF(2^{127}) \setminus \{0,1\} \times GF(2^{127}) \to GF(2^{127})\}$ *with* $g_k(m) = k \cdot m$ *for* $k \in GF(2^{127}) \setminus \{0,1\}$. *Then* G_{127} *is* $\frac{1}{2^{127}-2}$*-AXU.*

Proof
Each $k \in \mathrm{GF}(2^{127}) \setminus \{0, 1\}$ is a primitive element of $\mathrm{GF}(2^{127})$, since $2^{127} - 1$ is prime. Hence, the probability for $y = 0$ is always zero. So, for all $m, m', y \in \mathrm{GF}(2^{127})$ with $m \neq m'$ we have

$$\Pr_{g_k \in G_{127}}[g_k(m) \oplus g_k(m') = y] = \Pr_{k \in \mathrm{GF}(2^{127}) \setminus \{0,1\}}[m \cdot k + m' \cdot k = y] \leq \frac{1}{2^{127} - 2}.$$

□

The proof of Lemma 3 implies that there exists no such thing as weak keys Γ_X-AES. Other universal hash functions based on multiplication in $\mathrm{GF}(2^{128})$, e.g., Ghash [38]. Note,

$$3 \cdot 5 \cdot 17 \cdot 257 \cdot 641 \cdot 65537 \cdot 274177 \cdot 6700417 \cdot 67280421310721 = 2^{128} - 1.$$

This observation implies that $2^{128} - 1$ is not a prime, which again implies that $\mathrm{GF}(2^{128})$ has non trivial subfields. Let S a non trivial subfield of $\mathrm{GF}(2^{128})$. Then there is at least one element $e \in \mathrm{GF}(2^{128})$ that is a primitive element of S. The choice of e would imply that the probability that two distinct messages $m, m' \in \mathrm{GF}(2^{128})$ collide is at least $1/|S|$. Since $|S| < \mathrm{GF}(2^{128})$, we can call e a weak key.

This simple observation implies that GMAC [38] – which is based on Ghash – has more weak keys as one would naively expect, namely elements of $\mathrm{GF}(2^2)$. These weak keys are given by all elements of non trivial subgroups of $\mathrm{GF}(2^{128})$.

Lemma 4 (Block chaining property)
Let $H^i = \{h_k^i : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n\}$ be ϵ-AXU hash function families with $i = 0, ..., b$. Then the hash function family $H_c = \{h_k : \{0,1\}^n \times \{0,1\}^{nb} \to \{0,1\}^n\}$ defined as follows is $b\epsilon$-AXU:

$$c_0 = h_k^0(a), c_i = h_k^i(m_i \oplus c_{i-1}) h_k(m) = c_b.$$

Proof
For all $y, a, k, m_i, m_i' \in \{0,1\}^n$ with $i = 1, ..., b$ define $x_i = m_i \oplus c_{i+1}$. Assume that $m_1 \neq m_1'$. Then we know that $h_k^0(a) \oplus m_1 \neq h_k^0(a) \oplus m_1')$. From the definition of ϵ-AXU, it follows that

$$\Pr[h_k^1(h_k^0(a) \oplus m_1) \oplus h_k^1(h_k^0(a) \oplus m_1') = y] \leq \epsilon.$$

This gives us a new ϵ-AXU hash function which we write as $\hat{h}_k^1 = h_k^1 \circ h_k^0$. From Lemma 5 follows that for any pair $(m_1, m_2) \neq (m_1, m_2)$

$$\Pr[h_k^2(\hat{h}_k^1(m_1) \oplus m_2) \oplus h_k^2(\hat{h}_k^1(m_1') \oplus m_2') = y] \leq 2\epsilon.$$

The new hash function we defined by $\hat{h}_k^2 = h_k^2 \circ \hat{h}_k^1$ is a 2ϵ-AXU hash function. Lets denote $\hat{h}_k^j = h_k^j \circ \hat{h}_k^{j-1}$ for $j = 3, ..., b$. Applying Lemma 5 to every $\hat{h}_k^j$ shows that $\hat{h}_k^j$ is a $j\epsilon$-AXU hash function. While h_k is equivalent to $\hat{h}_k^b$, it follows that H_c is a family of $b\epsilon$-AXU hash functions. □

Lemma 5 (Chaining combination property)
Let $H_1 = \{h_1 : A \to B\}$ be ϵ_1-AXU and $H_2 = \{h_2 : B \to C\}$ be ϵ_2-AXU. Let $H = \{h : A \times B \to C\}$ be a hash function family with $h(a,b) = h_2(h_1(a) \oplus b)$ for $a \in A$ and $b \in B$, $h_1 \in H_1$ and $h_2 \in H_2$. Then H is ϵ-AXU with $\epsilon = \epsilon_1 + \epsilon_2$.

Proof
The value ϵ is the maximum probability of the event `Equal` defined as $h(a,b) \oplus h(a',b') = y \oplus y'$ for any $(a,b),(a',b') \in A \times B$ with $(a,b) \neq (a',b')$ and any $y, y' \in C$. Lets define $x = h_1(a) \oplus b$ and $x' = h_1(a') \oplus b'$. We distinguish the two cases $y \oplus y' = 0$ and $y \oplus y' \neq 0$

Case $y \oplus y' = 0$:

$$\begin{aligned}
&\Pr_{h\in H}[h(a,b) \oplus h(a',b') = 0] \\
&= \Pr_{\substack{h_1\in H_1\\ h_2\in H_2}}[h_2(\underbrace{h_1(a) \oplus b}_{x}) \oplus h_2(\underbrace{h_1(a') \oplus b'}_{x'}) = 0] \\
&= \Pr_{h_2\in H_2}[h_2(x) = h_2(x') \mid x = x'] \cdot \Pr[x = x'] \\
&\quad + \Pr_{h_2\in H_2}[h_2(x) = h_2(x') \mid x \neq x'] \cdot \Pr[x \neq x'] \\
&\leq \Pr[x = x'] + \Pr_{h_2\in H_2}[h_2(x) = h_2(x') \mid x \neq x']
\end{aligned}$$

From the definition of ϵ-AXU we know that $\Pr_{h_2\in H_2}[h_2(x) = h_2(x') \mid x \neq x'] \leq \epsilon_2$. If $a = a'$ then $b \neq b'$ since $(a,b) \neq (a',b')$. Hence, $\Pr[x = x'] = 0$ for $a = a'$. For $a \neq a$ we define $d = b \oplus b'$. Then for any $h_1 \in H$

$$\Pr[x = x'] = \Pr[h_1(a) \oplus h_1(a') = d].$$

From the definition of ϵ-AXU follows, that $\Pr[x = x'] \leq \epsilon_1$ for any $a \neq a'$. Therefore, the probability that two authentication tags collide is at most

$$\Pr_{h\in H}[h(a,b) \oplus h(a',b') = 0] \leq \epsilon_1 + \epsilon_2.$$

Case $z = y \oplus y' \neq 0$:
In this case, the event `Equal` occurs only if $x \neq x'$ and $h_2(x) \oplus h_2(x') = z$, since

$$\Pr_{h_2\in H_2}[h_2(x) \neq h_2(x') \mid x = x'] = 0.$$

Then,

$$\Pr_{h\in H}[h(a,b) \oplus h(a',b') = z] = \Pr_{h_2\in H_2}[h_2(x) \oplus h_2(x') = z \mid x \neq x'] \cdot \Pr[x \neq x'].$$

From the definition of ϵ-AXU follows that

$$\Pr_{h_2\in H_2}[h_2(x) \oplus h_2(x') = z \mid x \neq x'] \leq \epsilon_2.$$

Therefore,

$$\Pr_{h\in H}[h(a,b) \oplus h(a',b') = z] \leq \epsilon_2, \qquad z \neq 0.$$

The $\Pr_{h\in H}[h(a,b) \oplus h(a',b') = y \oplus y']$ is at most the maximum over the two cases $y \oplus y = 0$ and $y \oplus y \neq 0$, which is $\epsilon_1 + \epsilon_2$. □

6 Performance Analysis of Γ_X-AES

This section gives software performance benchmarks of Γ_X-AES. All measurement results are based on the real time clock (RTC). We obtained the performance benchmarks by measuring 2000 times the desired target function and then computing the median of those results.

Table 2. Benchmark results in *cycles per byte* on Intel Core i5 M540 2.53 GHz; OS: Linux 3.0.0-1; Compiler: GCC-4.61

Bytes	Γ_X**-AES** (Gamma-MAC)	**Poly1305-AES**† [5] (WCS-MAC)	**VMAC** [36] (WCS-MAC)	**AES-128** [19] (Ctr mode)
4	191	735	99.8	103
10	76.4	276	39.7	41.2
16	58	167.2	25	16.3
40	27.3	74.5	10.4	19.2
128	16.2	28.7	3.5	11.62
256	13.3	17.3	2.2	11.3
576	11.6	10.1	1.5	11.1
1000	10.9	8.5	1.2	11.3
1300	10.9	7.7	1.4	11.3
1500	10.7	7.5	1	11.2
4096	10.4	6.2	0.9	11
10000	10.3	N/A	1	11
16384	10.3	N/A	0.8	11.1
32768	10.3	N/A	0.8	11.1

† Benchmarks are taken from [6] (Median of Section *A4444–*).

Target Platform. The benchmarking took place on a 64-bit Intel Core i5 M540 2.53 GHz computer. All the software for the benchmarking was written either in C or assembler and compiled with the GNU C compiler (gcc) version 4.6.1 using the optimization flag `-O3`.

Implementation Remarks. The Γ_X-AES implementation for the benchmarking is based on 1) the MPFQ library from Gaudry and Thomé[2] which performs a GF(2^{127}) multiplication in about 9.1 cpb and 2) the AES-128 eSTREAM implementation from Bernstein and Schwabe[3] using optimization techniques from [7].

Results. Table 2 contains the results of our performance benchmarking. The table shows that Γ_X-AES outperforms AES-128 in counter mode for messages longer than 1000 bytes. Hence, Γ_{AES} would be faster than any variation of CBC-MAC based on AES-128. Note, that all our measurements were done on the specified 128-bit block size version of the MACs.

[2] `http://mpfq.gforge.inria.fr/mpfq-1.0-rc3.tar.gz`
[3] `http://cr.yp.to/streamciphers/timings/estreambench-20080905.tar.bz2`

7 Conclusion

In this work we presented Γ-MAC$[H, P]$, a new provable secure and stateless MAC scheme based on universal hash functions. In contrast to other universal MAC schemes, Γ-MAC$[H, P]$ needs only one single n-bit key. Most other universal MAC schemes like CWS-MAC and FH-MAC need at least two n-bit keys. Finally, we presented Γ_X-AES, an Γ-MAC$[H, P]$ instantiation based on Galois field multiplication and AES-128 similar to GMAC. Our implementation of Γ_X-AES outperforms AES-128 for messages that are longer than one kilobyte, on Intel Core i5 M540 2.53 GHz.

Acknowledgements. We would like to thank Emmanuel Thomé and Paul Zimmermann for the helpful support on the MPFQ library.

References

1. 3GPP. Specification of the 3GPP confidentiality and integrity algorithms. TS V.2.1.1, 3rd Generation Partnership Project (3GPP) (2001)
2. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
3. Bellare, M., Cash, D.: Pseudorandom Functions and Permutations Provably Secure against Related-Key Attacks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 666–684. Springer, Heidelberg (2010)
4. Bernstein, D.J.: Stronger Security Bounds for Wegman-Carter-Shoup Authenticators. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 164–180. Springer, Heidelberg (2005)
5. Bernstein, D.J.: The Poly1305-AES Message-Authentication Code. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer, Heidelberg (2005)
6. Bernstein, D.J.: Poly1305-AES speed tables (2009), http://cr.yp.to/mac/speed.html
7. Bernstein, D.J., Schwabe, P.: New AES Software Speed Records. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 322–336. Springer, Heidelberg (2008)
8. Biham, E., Dunkelman, O., Keller, N.: A Related-Key Rectangle Attack on the Full KASUMI. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 443–461. Springer, Heidelberg (2005)
9. Biryukov, A., Khovratovich, D.: Related-Key Cryptanalysis of the Full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer, Heidelberg (2009)
10. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and Related-Key Attack on the Full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
11. Black, J., Cochran, M.: MAC Reforgeability. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 345–362. Springer, Heidelberg (2009)

12. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: Fast and Secure Message Authentication. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 216–233. Springer, Heidelberg (1999)
13. Black, J., Rogaway, P.: CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 197–215. Springer, Heidelberg (2000)
14. Black, J., Rogaway, P.: A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 384–397. Springer, Heidelberg (2002)
15. Boesgaard, M., Christensen, T., Zenner, E.: Badger – A Fast and Provably Secure MAC. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 176–191. Springer, Heidelberg (2005)
16. Boesgaard, M., Scavenius, O., Pedersen, T., Christensen, T., Zenner, E.: Badger - A Fast and Provably Secure MAC. Cryptology ePrint Archive, Report 2004/319 (2004), http://eprint.iacr.org/
17. Borisov, N., Goldberg, I., Wagner, D.: Intercepting Mobile Communications: The Insecurity of 802.11. In: MOBICOM, pp. 180–189 (2001)
18. Brassard, G.: On Computationally Secure Authentication Tags Requiring Short Secret Shared Keys. In: McCurley, K.S., Ziegler, C.D. (eds.) CRYPTO 1982. LNCS, vol. 1440, pp. 79–86. Springer, Heidelberg (1982)
19. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. NIST AES Homepage (1999)
20. Dai, W., Krovetz, T.: VHASH Security. Cryptology ePrint Archive, Report 2007/338 (2007), http://eprint.iacr.org/
21. Dunkelman, O., Fleischmann, E., Gorski, M., Lucks, S.: Related-Key Rectangle Attack of the Full HAS-160 Encryption Mode. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT 2009. LNCS, vol. 5922, pp. 157–168. Springer, Heidelberg (2009)
22. Dunkelman, O., Keller, N., Shamir, A.: A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 393–410. Springer, Heidelberg (2010)
23. Jaulmes, É., Joux, A., Valette, F.: On the Security of Randomized CBC-MAC Beyond the Birthday Paradox Limit: A New Construction. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 237–251. Springer, Heidelberg (2002)
24. Etzel, M., Patel, S., Ramzan, Z.: SQUARE HASH: Fast Message Authentication via Optimized Universal Hash Functions. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 234–251. Springer, Heidelberg (1999)
25. Fleischmann, E., Gorski, M., Lucks, S.: Memoryless Related-Key Boomerang Attack on the Full Tiger Block Cipher. In: Bao, F., Li, H., Wang, G. (eds.) ISPEC 2009. LNCS, vol. 5451, pp. 298–309. Springer, Heidelberg (2009)
26. Gilbert, E.N., MacWilliams, F.J., Sloane, N.J.A.: Codes which detect deception. Bell System Tech. J. 53, 405–424 (1974)
27. Halevi, S., Krawczyk, H.: MMH: Software Message Authentication in the Gbit/Second Rates. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 172–189. Springer, Heidelberg (1997)
28. Handschuh, H., Preneel, B.: Key-Recovery Attacks on Universal Hash Function Based MAC Algorithms. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 144–161. Springer, Heidelberg (2008)
29. Hotz, G.: Console Hacking 2010 - PS3 Epic Fail. In: 27th Chaos Communications Congress (2010), http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf

30. Iwata, T., Kurosawa, K.: OMAC: One-Key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg (2003)
31. Ko, Y., Lee, C., Hong, S., Lee, S.: Related Key Differential Cryptanalysis of Full-Round SPECTR-H64 and CIKS-1. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 137–148. Springer, Heidelberg (2004)
32. Kohno, T.: Attacking and Repairing the WinZip Encryption Scheme. In: ACM Conference on Computer and Communications Security, pp. 72–81 (2004)
33. Kohno, T., Viega, J., Whiting, D.: CWC: A High-Performance Conventional Authenticated Encryption Mode. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 408–426. Springer, Heidelberg (2004)
34. Koo, B., Yeom, Y., Song, J.H.: Related-Key Boomerang Attack on Block Cipher SQUARE. IEICE Transactions 94-A(1), 3–9 (2011)
35. Krawczyk, H.: LFSR-Based Hashing and Authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994)
36. Krovetz, T.: Message Authentication on 64-Bit Architectures. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 327–341. Springer, Heidelberg (2007)
37. Kurosawa, K., Iwata, T.: TMAC: Two-Key CBC MAC. IEICE Transactions 87-A(1), 46–52 (2004)
38. 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)
39. National Institute of Standards and Technology. FIPS PUB 113: Standard for Computer Data Authentication. National Institute for Standards and Technology, Gaithersburg, MD, USA (May 1985)
40. Petrank, E., Rackoff, C.: CBC MAC for Real-Time Data Sources. J. Cryptology 13(3), 315–338 (2000)
41. Rogaway, P.: Bucket Hashing and Its Application to Fast Message Authentication. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 29–42. Springer, Heidelberg (1995)
42. Sabin, T.: Vulnerability in Windows NT's SYSKEY encryption. BindView Security Advisory (1999), http://marc.info/?l=ntbugtraq&m=94537191024690&w=4
43. Shoup, V.: On Fast and Provably Secure Message Authentication Based on Universal Hashing. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 313–328. Springer, Heidelberg (1996)
44. Stinson, D.R.: Universal Hashing and Authentication Codes. Des. Codes Cryptography 4(4), 369–380 (1994)
45. Wegman, M.N., Lawrence Carter, J.: New hash functions and their use in authentication and set equality. JCSSBM 22(3), 265–279 (1981)
46. Wu, H.: The Misuse of RC4 in Microsoft Word and Excel. Cryptology ePrint Archive, Report 2005/007 (2005), http://eprint.iacr.org/

New Universal Hash Functions

Aysajan Abidin and Jan-Åke Larsson

Department of Electrical Engineering,
Linköping University, SE-581 83 Linköping, Sweden
aysajan@isy.liu.se, jan-ake.larsson@liu.se

Abstract. Universal hash functions are important building blocks for unconditionally secure message authentication codes. In this paper, we present a new construction of a class of ϵ-Almost Strongly Universal$_2$ hash functions with much smaller description (or key) length than the Wegman-Carter construction. Unlike some other constructions, our new construction has a very short key length and a security parameter ϵ that is independent of the message length, which makes it suitable for authentication in practical applications such as Quantum Cryptography.

Keywords: Universal hash functions, ϵ-Almost Strongly Universal hash functions, authentication, Quantum Cryptography.

1 Introduction

Universal hash functions were first introduced by Wegman and Carter [7] in 1979, and since then they have been extensively studied. They are used in diverse cryptographic tasks such as unconditionally secure authentication, error-correction and randomness extraction (or privacy amplification, within Quantum Cryptography). Over the years, various Universal hash function families are constructed by Wegman and Carter, Stinson, and others [3,4,6,13,14,17,20–24]. The important properties are the description length (key consumption), the security parameter, and the computational efficiency, more on this below.

This paper addresses a new construction of Universal hash functions. In particular, we present a new construction of ϵ-Almost Strongly Universal$_2$ (ϵ-ASU$_2$) hash functions, that not only have small description length but also a security parameter ϵ that is independent of the message length. The construction combines the LFSR-based hashing proposed by Krawczyk in [13] with the composition theorem by Stinson in [20] for constructing Universal hash functions. Given its properties, the new construction is also computationally efficient.

1.1 Universal Hash Function Families

First, let us recall the definitions of Universal and ϵ-ASU$_2$ hash functions and the composition theorem for Universal hash functions.

Definition 1 (Universal$_2$ hash functions). *Let $\mathcal{M}$ and $\mathcal{T}$ be finite sets. A class $\mathcal{H}$ of hash functions from $\mathcal{M}$ to $\mathcal{T}$ is **Universal$_2$ (U_2)** if there exist at*

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 99–108, 2012.

most $|\mathcal{H}|/|\mathcal{T}|$ hash functions $h \in \mathcal{H}$ such that $h(m_1) = h(m_2)$ for any two distinct $m_1, m_2 \in \mathcal{M}$.

*If there are at most $\epsilon|\mathcal{H}|$ hash functions instead, the class $\mathcal{H}$ is ϵ-**Almost Universal$_2$** (ϵ-**AU$_2$**).*

Definition 2 (XOR Universal$_2$ hash functions). *Let $\mathcal{M}$ and $\mathcal{T}$ be as before. A class $\mathcal{H}$ of hash functions from $\mathcal{M}$ to $\mathcal{T}$ is **XOR Universal$_2$** (**XU$_2$**) if there exists at most $|\mathcal{H}|/|\mathcal{T}|$ hash functions $h \in \mathcal{H}$ such that $h(m_1) = h(m_2) \oplus t$ for any two distinct $m_1, m_2 \in \mathcal{M}$ and any $t \in \mathcal{T}$.*

*If there are at most $\epsilon|\mathcal{H}|$ hash functions instead, the class $\mathcal{H}$ is ϵ-**Almost XOR Universal$_2$** (ϵ-**AXU$_2$**).*

Definition 3 (Strongly Universal$_2$ hash functions). *Let $\mathcal{M}$ and $\mathcal{T}$ be as before. A class $\mathcal{H}$ of hash functions from $\mathcal{M}$ to $\mathcal{T}$ is **Strongly Universal$_2$** (**SU$_2$**) if the following two conditions are satisfied:*

(a) *The number of hash functions in $\mathcal{H}$ that takes an arbitrary $m_1 \in \mathcal{M}$ to an arbitrary $t_1 \in \mathcal{T}$ is exactly $|\mathcal{H}|/|\mathcal{T}|$.*
(b) *The fraction of those functions that also takes an arbitrary $m_2 \neq m_1$ in $\mathcal{M}$ to an arbitrary $t_2 \in \mathcal{T}$ (possibly equal to t_1) is $1/|\mathcal{T}|$.*

*If the fraction in (b) instead is at most ϵ, the class $\mathcal{H}$ is ϵ-**Almost Strongly Universal$_2$** (ϵ-**ASU$_2$**).*

Note that $\epsilon \geq 1/|\mathcal{T}|$ [21] so that SU$_2$ hash functions are the optimal case, corresponding to $1/|\mathcal{T}|$-ASU$_2$ hash functions.

There are several ways to construct classes of ϵ-ASU$_2$ hash functions, and in this paper we will use the following theorem from [20].

Theorem 1 (Composition). *Let $\mathcal{F}$ be a set of ϵ_1-AU_2 hash functions from $\mathcal{M} \to \mathcal{Z}$, and let $\mathcal{G}$ be a set of ϵ_2-ASU_2 hash functions from $\mathcal{Z} \to \mathcal{T}$. Then, $\mathcal{H} = \mathcal{G} \circ \mathcal{F}$ is an ϵ-ASU_2 hash function family from $\mathcal{M} \to \mathcal{T}$ with $\epsilon = \epsilon_1 + \epsilon_2$.*

We will also use ideas from [13,14], in which an ϵ-AXU$_2$ family is composed with a one-time pad, resulting in an ϵ-ASU$_2$ family (note that the above theorem does not apply). The resulting family has a security parameter ϵ that depends on the message length. In this paper, we will use a different approach that enables use of the theorem, and keeps $|\mathcal{H}|$ small while giving a security parameter ϵ that only depends on the tag length, not the message length.

1.2 Information-Theoretically Secure Authentication

The class of ϵ-ASU$_2$ hash functions can straightforwardly be applied for information-theoretically secure message authentication. In this scenario, two legitimate users (Alice and Bob) share a secret key k long enough to identify a hash function h_k in a family of ϵ-ASU$_2$ hash functions. When Alice wants to send a message m to Bob, she computes $t = h_k(m)$ and sends it along with m. Upon receiving m and t, Bob checks the authenticity of m by computing $h_k(m)$

using his share of the key and comparing it with t. If $h_k(m)$ and t are identical, then Bob accepts m as authentic; otherwise, he rejects it.

Now, if an adversary tries to impersonate Alice and sends m' without knowing the key k, or h_k, the best he/she can do is to guess the correct tag for m'. The probability of success in this case is $P_1 = 1/|\mathcal{T}|$. If the adversary intercepts a message-tag pair (m, t) from Alice and substitutes m with m', then the probability P_2 of guessing the correct tag t' for m' increases somewhat but is bounded by ϵ ($\geq 1/|\mathcal{T}|$). In other words, even seeing a valid message-tag pair does not increase the adversary's success probability above ϵ. Therefore, by using a family of ϵ-ASU$_2$ hash functions with suitably chosen ϵ, one can achieve information-theoretically secure message authentication.

In addition to requiring ϵ to be small, practical applications require also the length l of the key k identifying a hash function in the family of ϵ-ASU$_2$ hash functions to be as small as possible. This latter requirement is especially important in Quantum Cryptography (QC).

1.3 Application to Authentication in Quantum Cryptography

Quantum Cryptography (QC), also known as Quantum Key Distribution (QKD), is a key agreement technique based on the laws of quantum mechanics. The users first exchange quantum signals over a so-called quantum channel to generate a raw key by measuring the quantum signals. Then, they extract a common secret key from the raw key by performing a joint post-processing by communicating on an immutable public channel. The first QKD protocol known as BB84 was proposed by Bennett and Brassard in 1984 [2].

QKD is proven to be information-theoretically secure, provided that the public channel is immutable; see, for example, [19]. In the case that the public channel is not immutable (not authentic), QKD can easily be broken by a man-in-the-middle attack. Therefore, ensuring the authenticity of the public channel is a must. More specifically, the adversary must not be able to insert or modify the classical messages exchanged over the public channel between the legitimate users during the post-processing phase of the QKD protocol. Also, to guarantee information-theoretic security of QKD the authentication used must be information-theoretically secure.

This is achieved via ϵ-ASU$_2$ hashing, and thus needs shared secret key. In the first round the users must use pre-shared secret key, which should be long enough to authenticate the classical messages in the round. In the following rounds, key generated in the previous rounds must be used. Hence, the key-consumption rate of the authentication protocol used in QKD directly influences the key output rate. Because of this, one needs an authentication with small key-consumption rate. Furthermore, in QKD very long messages need to be authenticated, so it is desirable to have a scheme where it is simple to do this, without changing parameters of the communication protocol. Thus, there is a need for a hash function family that is small but still has a security parameter ϵ that only depends on the tag length, not the message length.

1.4 Lower Bounds

There are lower bounds on the description length (or key length) for ϵ-ASU$_2$ hash functions derived by Stinson [20], Kabatiankii *et al.* [12], Gemmel and Naor [9], and Nguyen and Roscoe [16]. In [16], the authors provided new combinatorial bounds that are tighter than the other bounds for the key length. They also identified a value for ϵ that represents a threshold in the behaviour of the various bounds and classified different lower bounds in relation to the threshold value of ϵ. Here, we only recall the lower bound by Stinson in [20]. Interested readers may refer to the above references for details of the other bounds.

Theorem 2 (Lower bound for ϵ-ASU$_2$ hash function families [20]). *If there exists an ϵ-ASU$_2$ family $\mathcal{H}$ of hash functions from $\mathcal{M}$ to $\mathcal{T}$, then*

$$|\mathcal{H}| \geq \frac{|\mathcal{M}|(|\mathcal{T}|-1)^2}{|\mathcal{T}|\epsilon(|\mathcal{M}|-1)+|\mathcal{T}|-|\mathcal{M}|}+1. \tag{1}$$

The proof can be found in [20]. In the SU$_2$ case, this simplifies to

$$|\mathcal{H}| \geq |\mathcal{M}|(|\mathcal{T}|-1)+1. \tag{2}$$

Otherwise, if $|\mathcal{M}| \gg |\mathcal{T}|$ the bound simplifies to (in terms of key length)

$$\log|\mathcal{H}| \geq 2\log(|\mathcal{T}|-1)-\log(\epsilon|\mathcal{T}|-1)+1. \tag{3}$$

Here and below "log" denotes the base 2 logarithm. If in addition $\epsilon = c/|\mathcal{T}|$ for some constant c and $|\mathcal{T}|$ is large, the right-hand side is close to $2\log|\mathcal{T}|$. If one allows ϵ to increase when $|\mathcal{M}|$ increases, the bounds decrease which makes it easier to approach $2\log|\mathcal{T}|$, as we shall see below.

1.5 Comparison of Some Existing Constructions

Now, let us briefly compare the key length and security parameter ϵ of a few constructions of ϵ-ASU$_2$ hash function families. In Table 1, the value of ϵ and the key length for five different constructions are listed for comparison. One can find a more detailed overview of various constructions of ϵ-ASU$_2$ hash functions by different authors in Refs. [1,17].

As can be seen from the table, the constructions by Wegman-Carter and Bierbrauer *et al.* have $\epsilon = 2/|\mathcal{T}|$ while the others have values of ϵ that depend on the message length either logarithmically (Stinson) or linearly (den Boer and Krawczyk). In terms of key length, den Boer's construction is the best followed by Krawczyk, having key lengths $2\log|\mathcal{T}|$ and $3\log|\mathcal{T}|+1$, respectively, and both are determined only by the tag length. The next good scheme in terms of key length is the construction by Bierbrauer *et al.,* for which the key length $\approx 3\log|\mathcal{T}|+2\log\log|\mathcal{M}|$. The key length for the constructions by Stinson and Wegman-Carter are logarithmic in the message length, but the construction by Stinson consumes approximately a quarter of the key that is needed for the

Table 1. The key length and ϵ for different constructions. The key length for Bierbrauer *et al* is approximate because of the need to invert se^s in the construction. This involves the Lambert W function (see, e.g., [8]), whose asymptotics for large s gives the expression below.

Construction	ϵ	Key length
Wegman-Carter [24]	$2/\lvert\mathcal{T}\rvert$	$4(\log\lvert\mathcal{T}\rvert + \log\log\log\lvert\mathcal{M}\rvert)\log\log\lvert\mathcal{M}\rvert$
Stinson [20]	$(\log\log\lvert\mathcal{M}\rvert - \log\log\lvert\mathcal{T}\rvert + 1)/\lvert\mathcal{T}\rvert$	$(\log\log\lvert\mathcal{M}\rvert - \log\log\lvert\mathcal{T}\rvert + 2)\log\lvert\mathcal{T}\rvert$
den Boer [6]	$(\log\lvert\mathcal{M}\rvert/\log\lvert\mathcal{T}\rvert)/\lvert\mathcal{T}\rvert$	$2\log\lvert\mathcal{T}\rvert$
Bierbrauer *et al.* [3]	$2/\lvert\mathcal{T}\rvert$	$\approx 3\log\lvert\mathcal{T}\rvert + 2\log\log\lvert\mathcal{M}\rvert$
Krawczyk [13]	$(1 + 2\log\lvert\mathcal{M}\rvert)/\lvert\mathcal{T}\rvert$	$3\log\lvert\mathcal{T}\rvert + 1$

Wegman-Carter. As mentioned earlier, we aim for a construction with small key length and ϵ independent of message size.

There are also other constructions such as Bucket hashing by Rogaway [18], MMH (Multilinear Modular Hashing) by Halevi and Krawczyk [10], and UMAC by Black *et al.* [5]. All three are very fast but have some properties that make them undesirable from the point of view of this paper. Bucket hashing has a very long key and output and is only ϵ-AU_2 and so the hash output has to be further mapped by an (A)SU_2 hash function to make it ϵ-ASU_2. Another paper [11] proposes a bucket hashing scheme with small key, but this does not have fixed ϵ and still has a comparatively long output. MMH [10] and UMAC [5] are not economical in terms of key length; the key lengths are very large in comparison to the above schemes.

1.6 Our Contribution

In this paper, we use LFSR-based hashing [13] and compose with an SU_2 hash function family. This enables the composition theorem, and gives a new ϵ-ASU_2 hash function family. One particular choice of parameters in the construction gives $\epsilon = 2/|\mathcal{T}|$ just as in Wegman-Carter's initial construction, while retaining a small description length. This construction is suitable for use in authentication, especially in Quantum Cryptography because of its low key-consumption property and the small fixed ϵ. Also, the new construction is also computationally efficient because the LFSR can efficiently be implemented in both software and hardware; the subsequent SU_2 hash function family operates on a much shorter intermediate bitstring, and is therefore also comparatively efficient.

2 The New Construction

In this section, we present our new construction of an ϵ-ASU_2 hash function family. To do this we need the first step in the construction by Krawczyk, the LFSR-based hashing [13].

2.1 LFSR-Based Hashing

In [13], Krawczyk presented an elegant way of constructing ϵ-AU_2 hash functions. The basic idea is to use an LFSR with a short key, a secret initial string and a secret feedback polynomial, to generate a longer key that selects a hash function in an ϵ-AU_2 hash function family. This can be viewed as selecting a certain subset of the linear maps from binary vectors m in $\mathcal{M}$ to binary vectors t in $\mathcal{T}$.

The full set of linear maps from $\mathcal{M}$ to $\mathcal{T}$ was found to be an SU_2 hash function family already by Wegman and Carter in [7], there denoted $\mathcal{H}_3$. In matrix language, $\mathcal{H}_3$ consists of $\log|\mathcal{T}| \times \log|\mathcal{M}|$ binary matrices, so that the description length of the hash functions in $\mathcal{H}_3$ is $(\log|\mathcal{M}|)(\log|\mathcal{T}|)$, which makes it impractical. However, if the matrices are restricted to be Toeplitz matrices (constant on diagonals), then the corresponding set of hash functions is still $\mathrm{Universal}_2$, see [15]. The description length of the hash functions is now reduced to $\log|\mathcal{M}| + \log|\mathcal{T}| - 1$, since a Toeplitz matrix can be uniquely identified by the first column and the first row of the matrix.

With a further restriction on the Toeplitz matrix, it is possible to obtain an ϵ_1-AXU_2 hash function family with a much smaller description length. In particular, if the consecutive columns of the Toeplitz matrix are restricted to be the consecutive states of an LFSR of length $\log|\mathcal{T}|$, then the hash functions with these matrices form an ϵ_1-AXU_2 hash function family with $\epsilon_1 = (2\log|\mathcal{M}|)/|\mathcal{T}|$. The description length of the hash functions in this family is $2\log|\mathcal{T}|+1$, which is the sum of the length of the initial state and the feedback polynomial; see [13] for details.

Krawczyk's construction continues with a composition with a one-time pad. In the next section, we will take a different route and not use the XOR property of the family, but only the ϵ-AU_2 property. In Krawczyk's construction, as mentioned in the introduction, the composition of an ϵ_1-AXU_2 family with a one-time pad (of length $|\mathcal{T}|$) is an ϵ-ASU_2 family with $\epsilon = \epsilon_1 + 1/|\mathcal{T}|$ [13]. The one-time pad has length $\log|\mathcal{T}|$. Therefore, the construction by Krawczyk has $\epsilon = (1 + 2\log|\mathcal{M}|)/|\mathcal{T}|$ and the key length $3\log|\mathcal{T}|+1$, which is the sum of the length of the key for LFSR-based hash function and of the one-time pad. We note that the feedback polynomials used in the LFSR-based hashing are irreducible, so that the actual key length is slightly less than $3\log|\mathcal{T}|+1$, see [13,14] for details on usage and key length.

2.2 LFSR-Based Hashing Followed by an SU_2 Hash Function Family

Our goal is to construct an ϵ-ASU_2 hash function family from $\mathcal{M}$ to $\mathcal{T}$ with $\epsilon = 2/|\mathcal{T}|$ and with a small key length. To this end, we use LFSR-based hashing and the composition theorem. Recall that the composition theorem states that if $h = g \circ f$ is the composition of an ϵ_1-AU_2 hash function f from $\mathcal{M} \to \mathcal{Z}$ with an SU_2 hash function g from $\mathcal{Z} \to \mathcal{T}$, then h is ϵ-ASU_2 with $\epsilon = \epsilon_1 + 1/|\mathcal{T}|$ from $\mathcal{M} \to \mathcal{T}$. Also, if f is an LFSR-based hash function from $\mathcal{M} \to \mathcal{Z}$, then f is an ϵ_1-AU_2 with $\epsilon_1 = (2\log|\mathcal{M}|)/|\mathcal{Z}|$. Therefore, to make

$$\frac{2\log|\mathcal{M}|}{|\mathcal{Z}|} + \frac{1}{|\mathcal{T}|} = \frac{2}{|\mathcal{T}|}, \tag{4}$$

we need to have

$$\frac{2\log|\mathcal{M}|}{|\mathcal{Z}|} = \frac{1}{|\mathcal{T}|}, \tag{5}$$

which gives us

$$|\mathcal{Z}| = 2|\mathcal{T}|\log|\mathcal{M}|. \tag{6}$$

This gives the following construction: let $\mathcal{F}$ be a set of LFSR-based hash functions from $\mathcal{M} \to \mathcal{Z}$, where $\mathcal{Z}$ is an intermediate set of bit strings of length $\log|\mathcal{T}| + \log\log|\mathcal{M}| + 1$. From eqn. (5), we see that $\mathcal{F}$ is an ϵ-AU$_2$ hash function family with $\epsilon = (2\log|\mathcal{M}|)/|\mathcal{Z}| = 1/|\mathcal{T}|$. Let $\mathcal{G}$ be a set of SU$_2$ hash functions from $\mathcal{Z} \to \mathcal{T}$, and $\mathcal{H} = \mathcal{G} \circ \mathcal{F}$. Then, by the composition theorem, it follows that $\mathcal{H}$ is a family of ϵ-ASU$_2$ hash functions from $\mathcal{M} \to \mathcal{T}$ with

$$\epsilon = 2/|\mathcal{T}|. \tag{7}$$

As before, the family $\mathcal{F}$ of LFSR-based hash functions from $\mathcal{M} \to \mathcal{Z}$ has description length $l_{\mathcal{F}} = 2\log|\mathcal{Z}| + 1$. And since $\mathcal{Z}$ is a set of strings of length $\log|\mathcal{T}| + \log\log|\mathcal{M}| + 1$ we obtain $l_{\mathcal{F}} = 2\log|\mathcal{T}| + 2\log\log|\mathcal{M}| + 3$.

For the SU$_2$ family of hash functions $\mathcal{G}$, the shortest possible description length is slightly smaller than $\log|\mathcal{Z}| + \log|\mathcal{T}|$ because of the bound (2). The construction in Lemma 10 of Bierbrauer *et al.* [3] almost reaches this with a key length of exactly $\log|\mathcal{Z}| + \log|\mathcal{T}|$, which gives a description length of the SU$_2$ hash functions in $\mathcal{G}$ of $l_{\mathcal{G}} = 2\log|\mathcal{T}| + \log\log|\mathcal{M}| + 1$. Explicitly, let π be a linear map from $\mathcal{Z} \to \mathcal{T}$. Then, the family $\mathcal{G}$ of hash functions $g : \mathcal{Z} \to \mathcal{T}$ defined as $g_{z,t}(r) = \pi(zr) + t$, where $z, r \in \mathcal{Z}$ and $t \in \mathcal{T}$, is SU$_2$. This family works well as $\mathcal{G}$, but note that any SU$_2$ family with key length equal to the message (intermediate bit string) length plus the tag length would give the same total key length

$$l_{\mathcal{H}} = l_{\mathcal{F}} + l_{\mathcal{G}} = 4\log|\mathcal{T}| + 3\log\log|\mathcal{M}| + 4. \tag{8}$$

3 Comparison with Existing Constructions

Let us now compare the above construction with existing constructions in terms of the key length, security parameter, and performance. Table 2 lists the relevant data in terms of the key length and the security parameter ϵ.

As can be seen from the table, the new construction like the constructions by Wegman-Carter [24] and Bierbrauer *et al.* [3] has a fixed $\epsilon = 2/|\mathcal{T}|$, while the others have ϵ dependent logarithmically or linearly on the message length $\log|\mathcal{M}|$. In terms of the key length, our construction clearly consumes much less key than the constructions by Wegman-Carter [24] and Stinson [20], but not as little as the constructions by den Boer, Krawczyk, and Bierbrauer. The construction by den Boer has the lowest key length $2\log|\mathcal{T}|$ at the cost of an increase in ϵ.

Table 2. The key length and ϵ for the new and the existing constructions. Approximations as before.

Construction	ϵ	Key length
Wegman-Carter [24]	$2/\lvert\mathcal{T}\rvert$	$4(\log\lvert\mathcal{T}\rvert + \log\log\log\lvert\mathcal{M}\rvert)\log\log\lvert\mathcal{M}\rvert$
Stinson [20]	$(\log\log\lvert\mathcal{M}\rvert - \log\log\lvert\mathcal{T}\rvert + 1)/\lvert\mathcal{T}\rvert$	$(\log\log\lvert\mathcal{M}\rvert - \log\log\lvert\mathcal{T}\rvert + 2)\log\lvert\mathcal{T}\rvert$
den Boer [6]	$(\log\lvert\mathcal{M}\rvert/\log\lvert\mathcal{T}\rvert)/\lvert\mathcal{T}\rvert$	$2\log\lvert\mathcal{T}\rvert$
Bierbrauer *et al.* [3]	$2/\lvert\mathcal{T}\rvert$	$\approx 3\log\lvert\mathcal{T}\rvert + 2\log\log\lvert\mathcal{M}\rvert$
Krawczyk [13]	$(1 + 2\log\lvert\mathcal{M}\rvert)/\lvert\mathcal{T}\rvert$	$3\log\lvert\mathcal{T}\rvert + 1$
This construction	$2/\lvert\mathcal{T}\rvert$	$4\log\lvert\mathcal{T}\rvert + 3\log\log\lvert\mathcal{M}\rvert + 4$

Another way to compare the schemes is to fix the security parameter ϵ, and from that and the message length $\log\lvert\mathcal{M}\rvert$ determine tag length $\log\lvert\mathcal{T}\rvert$ and key length. This is done in Table 3, but only for the four last alternatives of Table 2. As we can see from the table, the tag length does not depend on $\log\lvert\mathcal{M}\rvert$ for Bierbrauer *et al.* and the present scheme, while it increases when the message size increases for den Boer [6] and Krawczyk [13]. In terms of key length dependence on $\log\lvert\mathcal{M}\rvert$, the constructions by den Boer [6] and Bierbrauer *et al.* are somewhat better than Krawczyk [13] and the current constructions.

Table 3. The key length and tag length, given ϵ and $\lvert\mathcal{M}\rvert$. Here, also the entries for den Boer are approximate; an approximation of the inverse to $\lvert\mathcal{T}\rvert\log\lvert\mathcal{T}\rvert$ again involves the asymptotics of the Lambert W function.

Construction	$\log\lvert\mathcal{T}\rvert$	Key length
den Boer [6]	$\approx -\log\epsilon + \log\log\lvert\mathcal{M}\rvert$	$\approx -2\log\epsilon + 2\log\log\lvert\mathcal{M}\rvert$
Bierbrauer *et al.* [3]	$-\log\epsilon + 1$	$\approx -3\log\epsilon + 2\log\log\lvert\mathcal{M}\rvert$
Krawczyk [13]	$-\log\epsilon + \log(1 + 2\log\lvert\mathcal{M}\rvert)$	$-3\log\epsilon + 3\log(1 + 2\log\lvert\mathcal{M}\rvert) + 1$
This construction	$-\log\epsilon + 1$	$-4\log\epsilon + 3\log\log\lvert\mathcal{M}\rvert + 8$

Finally, simplicity of use and setup and performance should be briefly addressed. It is simpler to aim for a given security if there is only one parameter to adjust, and this would be a benefit of the present construction and the one by Bierbrauer *et al.* [3]. If the security parameter ϵ is fixed, so is the tag length in these two. The other two need to change tag length when the message length changes.

In terms of performance, the present construction and the one by Krawczyk [13] seem to have an advantage, since both decrease the size of the long message by using an LFSR which can efficiently be implemented in hardware and software. The other two use modular arithmetic in larger fields, which is somewhat less efficient. After shortening the message, Krawczyk's construction uses an OTP, again using efficient binary arithmetic, while the construction in this

paper maps the intermediate short string into a tag by an SU_2 hash function. The difference between the two operations is not so large, since the length of the intermediate string is not so long in our construction. In all, the hash function family proposed in this paper compares well to the others in both cases, in that it is the only family that has both a simple relation between security parameter and construction parameters, *and* is efficient.

4 Conclusion

We have presented a simple new construction of an efficient ϵ-ASU_2 hash function family with small description length, for which the security parameter is independent of message length. The construction uses the idea of LFSR-based hashing together with Stinson's composition theorem for Universal hash function families. The resulting family has a key consumption that is logarithmic in the message length and linear in the tag length or logarithmic in the security parameter, with small (constant) coefficients. It is efficient, given that it requires a short key. These properties make our construction very suitable for information-theoretically secure authentication purposes in practical applications, especially in Quantum Cryptography.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable comments on an earlier version of the paper.

References

1. Atici, M., Stinson, D.R.: Universal Hashing and Multiple Authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 16–30. Springer, Heidelberg (1996)
2. Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. In: Proc. IEEE Int. Conf. Comput. Syst. Signal Process., Bangalore, India, pp. 175–179 (1984)
3. Bierbrauer, J., Johansson, T., Kabatianskii, G., Smeets, B.: On Families of Hash Functions via Geometric Codes and Concatenation. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 331–342. Springer, Heidelberg (1994)
4. Black, J.: Message authentication codes. Ph.D. thesis, University of California Davis, USA (2000)
5. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: Fast and Secure Message Authentication. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 216–233. Springer, Heidelberg (1999)
6. den Boer, B.: A simple and key-economical unconditional authentication scheme. J. Comp. Sec. 2, 65–72 (1993)
7. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. 18, 143–154 (1979)
8. Corless, R.M., Gonnet, G.H., Hare, D.E.G., Jeffrey, D.J., Knuth, D.E.: On the Lambert W function. Adv. Comput. Math. 5, 329–359 (1996)

9. Gemmell, P., Naor, M.: Codes for Interactive Authentication. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 355–367. Springer, Heidelberg (1994)
10. Halevi, S., Krawczyk, H.: MMH: Software Message Authentication in the Gbit/Second Rates. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 172–189. Springer, Heidelberg (1997)
11. Johansson, T.: Bucket Hashing with a Small Key Size. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 149–162. Springer, Heidelberg (1997)
12. Kabatianskii, G., Smeets, B.J.M., Johansson, T.: On the cardinality of systematic authentication codes via error-correcting codes. IEEE Trans. Inf. Theory 42, 566–578 (1996)
13. Krawczyk, H.: LFSR-Based Hashing and Authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994)
14. Krawczyk, H.: New Hash Functions for Message Authentication. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 301–310. Springer, Heidelberg (1995)
15. Mansour, Y., Nisan, N., Tiwari, P.: The computational complexity of universal hashing. In: Ortiz, H. (ed.) Proc. STOC 1990, pp. 235–243. ACM, New York (1990)
16. Nguyen, L.H., Roscoe, A.W.: A new bound for t-wise almost universal hash functions. IACR Cryptology ePrint Archive, Report 2009/153 (2009), http://eprint.iacr.org/2009/153
17. Preneel, B.: Analysis and design of cryptographic hash functions. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium (1993)
18. Rogaway, P.: Bucket Hashing and Its Application to Fast Message Authentication. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 29–42. Springer, Heidelberg (1995)
19. Shor, P.W., Preskill, J.: Simple proof of security of the bb84 quantum key distribution protocol. Phys. Rev. Lett. 85, 441–444 (2000)
20. Stinson, D.R.: Universal Hashing and Authentication Codes. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 74–85. Springer, Heidelberg (1992)
21. Stinson, D.R.: Combinatorial techniques for universal hashing. J. Comput. Syst. Sci. 48, 337–346 (1994)
22. Stinson, D.R.: On the connections between universal hashing, combinatorial designs and error-correcting codes. Congressus Numerantium 114, 7–27 (1996)
23. Stinson, D.R.: Universal hash families and the leftover hash lemma, and applications to cryptography and computing. J. Combin. Math. Combin. Comput. 42, 3–31 (2002)
24. Wegman, M.N., Carter, L.: New hash functions and their use in authentication and set equality. J. Comput. Syst. Sci. 22, 265–279 (1981)

Cryptanalysis of TWIS Block Cipher

Onur Koçak and Neşe Öztop

Institute of Applied Mathematics, Middle East Technical University, Turkey
{onur.kocak,noztop}@metu.edu.tr

Abstract. TWIS is a 128-bit lightweight block cipher that is proposed by Ojha et al. In this work, we analyze the security of the cipher against differential and impossible differential attacks. For the differential case, we mount a full-round attack on TWIS and recover 12 bits of the 32-bit final subkey with 2^{21} complexity. For the impossible differential, we present a distinguisher which can be extended to a key recovery attack. Also, we showed that the security of the cipher is only 54 bits instead of claimed 128 bits. Moreover, we introduce some observations that compromise the security of the cipher.

Keywords: TWIS, Lightweight Block Cipher, Differential Cryptanalysis, Impossible Differential Distinguisher.

1 Introduction

The pace of ubiquitous devices in daily life has been increased drastically in the last few years. As the usage increases, the privacy of the stored data and the security of the communication between these devices become questionable. The requirement for protection of data and communication makes the use of cryptographic algorithms inevitable. However, the standardized algorithms like AES [1] and SHA [2] or commonly used algorithms like Triple DES [3] and MD5 [4] are not suitable for constrained devices. Therefore, recently, new lightweight algorithms which need low power consumption and hardware area, like Present [5], KATAN/KTANTAN [6], DESL [7], Grain [8] and TWIS [9] are designed for such constrained environments.

TWIS is a 128-bit block cipher designed to be used in ubiquitous devices. The cipher, which is inspired by CLEFIA [10], is a 2-branch generalized Feistel Network of 10 rounds. There is no key recovery attack on this cipher up to the authors knowledge. The only analysis is done by Su et al. [11] in which n-round iterative differential distinguishers are presented. However, as the probability of the iterative distinguishers are 1, they cannot be extended to a differential attack to get information about the key.

In this paper, we analyze the security of TWIS block cipher against differential and impossible differential cryptanalysis. We mount a differential attack on full-round TWIS and recover 12 bits of the 32-bit final subkey with a complexity of 2^{21}. This is the first key recovery attack on TWIS. Also, we present a 9.5-round

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 109–121, 2012.

impossible distinguisher which can be extended to a key recovery attack. Furthermore, by making observations on the key schedule, we show that the cipher offers at most 54-bit security instead of claimed 128-bit. Besides, we mention the potential weaknesses due to the use of subkeys during the encryption and the choice of whitening subkeys. The paper is organized as follows. Section 2 gives a description of the round function and the key schedule of TWIS block cipher. In Section 3, a 10-round differential attack is presented. Impossible differential distinguisher is proposed in Section 4. Some observations on the algorithm are given in Section 5. Finally, Section 6 concludes the paper.

2 Description of TWIS Block Cipher

TWIS is a lightweight block cipher with 128-bit plaintext and key sizes each. Designers of the cipher are inspired by CLEFIA in order to design a lighter algorithm without compromising the security. The algorithm is a 2-branch generalized Feistel Network, running on 10 rounds. At each round, two 32-bit subkeys are used. The key is mixed with the plaintext inside the G-function. Details of the G-function is given in Section 2.1.

Round subkeys are generated via key scheduling algorithm. Key scheduling part can be viewed as an $NFSR$ which updates the content using an S-box and a round constant. Details of the key scheduling algorithm is given in Section 2.2.

Notation. The following notations are used throughout this paper:

- a $\oplus$ b: bitwise XOR of a and b
- a $\wedge$ b: bitwise AND of a and b
- $<<<\ i$: left rotation by i bits
- $>>>\ i$: right rotation by i bits
- ΔI: XOR difference between two inputs

Let $P = (P_0, P_1, P_2, P_3)$ and $C = (C_0, C_1, C_2, C_3)$ be the 128-bit plaintext and ciphertext respectively where P_i and C_i, $0 \leq i \leq 3$, are 32-bit words. Also, let RK_j be the 32-bit j^{th} subkey for $j = 0, .., 10$. Then, the encryption process can be summarized as in Algorithm 1. Likewise, Figure 2 shows the encryption schematically.

2.1 *G*-Function

G-function is the round function of TWIS block cipher. It provides confusion and diffusion between the branches. G-function takes three inputs of 32 bits; 32-bit subkey and 32-bit words from each two of the four branches, and outputs two 32-bit words. The G-function can be written as in Algorithm 2.

Algorithm 1. The Encryption Process of TWIS

$(T_0, T_1, T_2, T_3) = (P_0 \oplus RK_0, P_1, P_2, P_3 \oplus RK_1)$
for $i = 1$ **to** 10 **do**
  $(X_0, X_1) = G(RK_{i-1}, T_0, T_1)$
  $T_2 = X_0 \oplus T_2$
  $T_3 = X_1 \oplus T_3$
  $T_1 = T_1 <<< 8$
  $T_3 = T_3 >>> 1$
  $(T_0, T_1, T_2, T_3) = (T_2, T_3, T_0, T_1)$
  $(X_0, X_1) = G(RK_i, T_0, T_3)$
  $T_1 = X_0 \oplus T_1$
  $T_2 = X_1 \oplus T_2$
  $T_2 = T_2 >>> 1$
  $T_3 = T_3 <<< 8$
end for
$(C_0, C_1, C_2, C_3) = (T_0 \oplus RK_2, T_1, T_2, T_3 \oplus RK_3)$

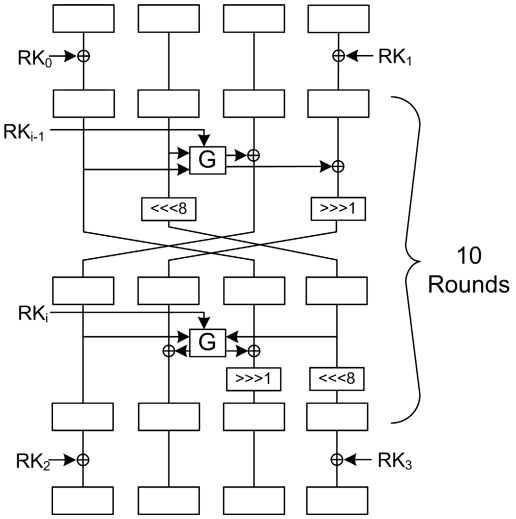


Fig. 1. Encryption Process

Algorithm 2. G-Function

$G(RK, X_0, X_1) = (Y_0, Y_1)$
  $Y_1 = X_1 \oplus F(RK, X_0)$
  $Y_0 = X_1$

***F*-Function.** F-function is the core of the G-function. Key mixing and confusion occurs within this function. F-function takes two 32-bit inputs, one of which is the subkey. It XORs the first parameter, the content of the corresponding branch, with the subkey and divides the resulting 32-bit word into four 8-bit words. Then, F-function applies a 6×8 S-box to each of the 8-bit words and swaps them. Finally, it concatenates four 8-bit words to form a 32-bit word. The F-function is formulated in Algorithm 3.

Algorithm 3. F-Function

$F(RK, Q)$
$Q = Q \oplus RK$
$Q = (Q_0, Q_1, Q_2, Q_3)$
$Q_0 = S(Q_0 \wedge 0x3f)$
$Q_1 = S(Q_1 \wedge 0x3f)$
$Q_2 = S(Q_2 \wedge 0x3f)$
$Q_3 = S(Q_3 \wedge 0x3f)$
$Q = (Q_2, Q_3, Q_0, Q_1)$

S-Box. The S-box used in the F-function is a 6×8 S-box which is given in Table 1. The first two bits of the 6-bit input determine the row, the remaining 4 bits determine the column of the table and the corresponding value is given as the output. For example $S(0x24) = 0xf7$.

Although the output space is larger than the input space, there are some inputs that are mapped to the same output, like $S(30) = S(15)$. This enables a non-zero difference to be mapped to zero difference which is a weakness that can be exploited to mount differential type attacks.

Table 1. S-Box of TWIS

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	90	49	d1	c6	2f	33	74	fb	95	6d	82	ea	0e	b0	a8	1c
1	28	d0	4b	92	5c	ee	85	b1	c4	0a	76	3d	63	f9	17	af
2	bf	bf	19	65	f7	7a	32	20	16	ce	e4	83	9d	5b	4c	d8
3	ee	99	2e	f8	d4	9b	0f	13	29	89	67	cd	71	dd	b6	f4

2.2 Key Schedule

The key schedule part of TWIS generates subkeys which are used in the F-functions. It produces 11 subkeys for the 10-round cipher. RK_0 and RK_1 are used as the initial whitening keys, while RK_2 and RK_3 are used as the final whitening

keys. Notice that, RK_1, RK_2 and RK_3 are used three times, RK_{10} is used once and the rest of the subkeys are used twice. The key scheduling algorithm uses the same S-box as in the F-function. In addition, it uses a diffusion matrix M to generate the subkeys from the master key which is given as

$$M = \begin{pmatrix} 0x01 & 0x02 & 0x04 & 0x06 \\ 0x02 & 0x01 & 0x06 & 0x04 \\ 0x04 & 0x06 & 0x01 & 0x02 \\ 0x06 & 0x04 & 0x02 & 0x01 \end{pmatrix}.$$

The key scheduling algorithm can be formulated as in Algorithm 4 and is shown in Figure 2.

Algorithm 4. The Key Scheduling Algorithm

$K = (K_1, K_2, \ldots, K_{16})$
for $i = 1$ **to** 11 **do**
  $K = K \lll 3$
  $K_i = S(K_i \wedge 0x3f)$
  $K_{15} = S(K_{15} \wedge 0x3f)$
  $K_{16} = K_{16} \oplus i$
  $RK_{i-1}^t = M \cdot (K_{13}K_{14}K_{15}K_{16})^t$
end for

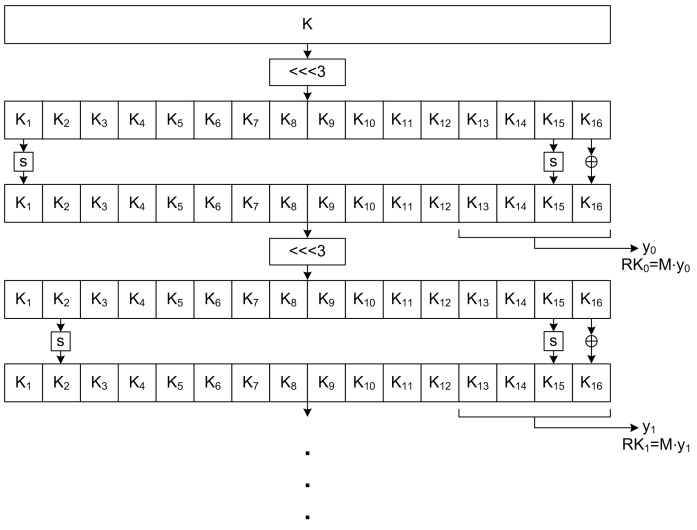


Fig. 2. Key Scheduling Algorithm

3 Differential Attack on TWIS

Differential cryptanalysis was introduced by Biham and Shamir [12] in 1990 and is one of the most effective techniques in block cipher cryptanalysis. It analyzes how the difference between two input values propagates after encryption of these inputs for some number of rounds. For TWIS block cipher, no differential analysis is given and it is left as a future work [9]. In [11], security of TWIS against differential cryptanalysis is evaluated by Su et al. and differential distinguishers for 10-round TWIS cipher are presented. In this section, we propose a key recovery attack on 10-round TWIS excluding the final key whitening. Our attack is based on a 9.5-round differential distinguisher which is explained in the following section.

3.1 9.5-Round Differential Characteristic

In order to construct a differential characteristic with high probability, we choose the differences utilizing the following properties:

Property 1. The first 2 bits of 8-bit input which enters the S-box have no effect on the output because of the bitwise AND operation with $0x3f$.

Property 2. The input differences $0x01$ and $0x25$ cause zero output differences with probability 2^{-5}.

Property 1 enables us to have 1-round differentials with probability 1. Also, using Property 2, the number of active S-boxes can be decreased.

The inputs of the F-function are the 1^{st} and the 3^{rd} 32-bit words of the data which are interchanging in the swap operation. There is no rotation operation applied on the 3^{rd} word and the rotation on the 1^{st} word is a 1-bit right rotation. Therefore, if we have 80000000_x as input difference in the 3^{rd} word, this difference will produce zero differences after the F-function with probability 1 during the next four rounds by the first property. We extend such a 4-round characteristic by adding 3 rounds to the beginning and 2.5 rounds to the end of it. The best characteristic that we found for TWIS has probability 2^{-18} and is given in Table 2. For simplicity, we use the alternative round function depicted in Figure 3. In Table 2, the values ΔI_i refer to the input differences of the corresponding round. The output differences are not given additionally as they are the input differences of the next round.

Notice that, in Table 2, the probability values of some rounds are marked with an asterisk(*) and these values are also relatively higher when considering the number of active S-boxes. The reason for high probability is that the cipher uses the same subkey for two consecutive G-functions and this makes these G-functions identical. To clarify, let x and $\bar{x}$ be two input values to G and y, $\bar{y}$ be the two corresponding output values. Then, if x and $\bar{x}$ are input to the next G-function which uses the same subkey, the outputs will again be y and $\bar{y}$. Hence, if an input pair with input difference Δx produces outputs with difference Δy

Table 2. 9.5-round Differential Characteristic

Rounds	ΔI_0	ΔI_1	ΔI_2	ΔI_3	# Active S-boxes	I/O Diff. for S-box	Probability
1	02000000_x	00000000_x	00000000_x	$0000A600_x$	1	$0x02 \rightarrow 0xA6$	2^{-4}
2	00000000_x	00000000_x	01000000_x	00000000_x	1	$0x01 \rightarrow 0x00$	2^{-5}
3	01000000_x	00000000_x	00000000_x	00000000_x	1	$0x01 \rightarrow 0x00$	1*
4	00000000_x	00000000_x	00800000_x	00000000_x	0	-	1
5	00800000_x	00000000_x	00000000_x	00000000_x	0	-	1
6	00000000_x	00000000_x	00400000_x	00000000_x	0	-	1
7	00400000_x	00000000_x	00000000_x	00000000_x	0	-	1
8	00000000_x	00000000_x	00200000_x	00000000_x	1	$0x20 \rightarrow 0x83$	2^{-4}
9	00200000_x	00000000_x	80000041_x	00000000_x	2	$0x20 \rightarrow 0x83$ $0x01 \rightarrow 0x00$	2^{-5}*
9.5	80000041_x	80000041_x	00100000_x	00000000_x	1	$0x01 \rightarrow 0x00$	1*
	80000041_x	00004180_x	80100041_x	$C0000020_x$	-	-	

with some probability p in G, then the same output difference Δy is produced with probability 1 when the input difference is Δx for the next G-function that uses the same subkey. Therefore, the probability of a differential characteristic that involves such G-functions is p instead of p^2. If each G-function were using different subkeys, the probability of the characteristic would be 2^{-32}.

3.2 10-Round Differential Attack

We perform a key-recovery attack on 10-round TWIS, excluding the final key whitening, by using the 9.5-round differential characteristic given in Section 3.1 and recover 12 bits of the last round subkey RK_{10}. Adding a half round to the end of the given 9.5-round differential characteristic and simply tracing the differences, we obtain the difference between ciphertext pairs as (80100041_x,$C00041A0_x$, $????????_x$, 00418000_x).

The attack proceeds as follows:

1. Take $N = c.2^{18}$ plaintext pairs $P^i = (P_0^i, P_1^i, P_2^i, P_3^i)$, $P^{i^*} = (P_0^{i^*}, P_1^{i^*}, P_2^{i^*}, P_3^{i^*})$ such that $P^i \oplus P^{i^*} = (02000000_x, 00000000_x, 00000000_x, 0000A600_x)$ and obtain their corresponding ciphertexts $C^i = (C_0^i, C_1^i, C_2^i, C_3^i)$, $C^{i^*} = (C_0^{i^*}, C_1^{i^*}, C_2^{i^*}, C_3^{i^*})$ by encrypting these plaintexts for 10 rounds of TWIS.
2. Check the first 64-bit and the last 32-bit ciphertext difference whether $C_0^i \oplus C_0^{i^*} = 80100041_x$, $C_1^i \oplus C_1^{i^*} = C00041A0_x$ and $C_3^i \oplus C_3^{i^*} = 00418000_x$ and keep the text pairs satisfying these equations.
3. As the input differences of the S-boxes in the 10^{th} round are $0x3f \cdot 80 = 0x0$, $0x3f \cdot 10 = 0x10$, $0x3f \cdot 00 = 0x0$ and $0x3f \cdot 41 = 0x01$, one can attack the 2^{nd} and 4^{th} 8-bit words of RK_{10}. However, since two bits of each word vanish after bitwise AND operation in the S-box, we can retrieve 12 bits of the subkey. Therefore, keep a counter for each possible value of the 12 bits of the subkey RK_{10} corresponding to the second and the fourth bytes.

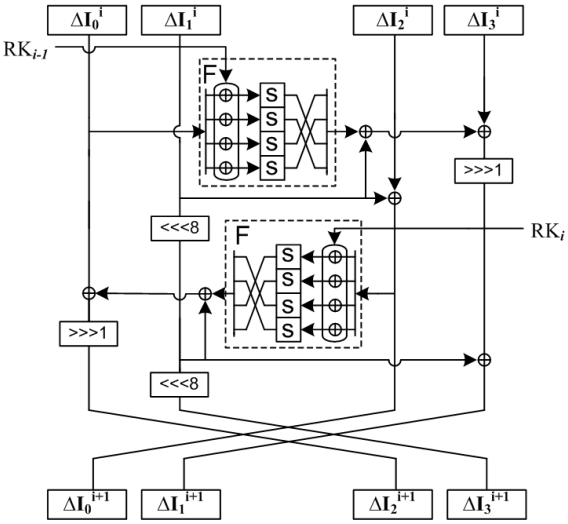


Fig. 3. Alternative Round Function

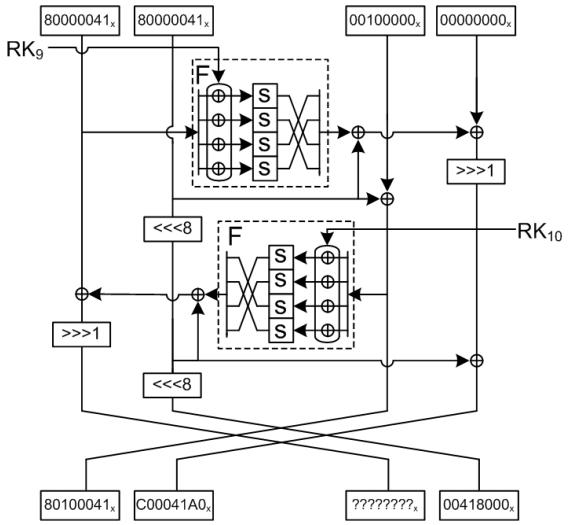


Fig. 4. Last Round of the Attack

4. Inputs of the last F-function are (C_0^i, RK_{10}) and (C_0^{i*}, RK_{10}). XOR of output difference of this F-function and $((00418000_x) >>> 8)$ should be equal to the XOR of 80000041_x and $(\Delta C_2^i <<< 1)$. So, for each pair of plaintexts and their corresponding ciphertexts (C^i, C^{i*}), increment the counter for the corresponding value of the subkey RK_{10} when the following equations holds:

$$F(C_0^i, RK_{10}) \oplus F(C_0^{i*}, RK_{10}) \oplus 00004180_x = 80000041_x \oplus (\Delta C_2^i <<< 1).$$

5. Adopt the key with the highest counter as the right key.

The signal to noise ratio S/N of the attack is calculated as 2^5. This value can be calculated from

$$S/N = \frac{2^k \cdot p}{\alpha \cdot \beta}$$

where k is the key bits we try to derive, p is the probability, α is the average count of the subkeys per counted plaintext pair and β is the ratio of the counted pairs to all pairs. As we search for the 12 bits of the final subkey, $k = 12$, and the probability is $p = 2^{-18}$. The expected number of suggested subkeys is $\alpha = 2$. The checking condition for the output of the F-function is 12 bits. So, S/N ratio can be computed as

$$S/N = \frac{2^{-18} \cdot 2^{12}}{2 \cdot 2^{-12}} = 2^5.$$

According to [12], about $c = 4$ right pairs is enough to uniquely determine the 12 bits of RK_{10}. Therefore, the number of required plaintext pairs is $N = 4 \cdot 2^{18} = 2^{20}$ and this makes the data complexity of the attack 2^{21} chosen plaintexts. Step (1) requires 2^{21} 10-round encryptions. After Step (2), there remains $2^{20} \cdot 2^{-18} = 2^2$ right pairs. Step (3) requires 2^{12} counters. For Step (4), $2^2 \cdot 2 \cdot \frac{1}{2}$ 1-round computations are required which can be ignored. Hence, time complexity of this attack is 2^{21} 10-round encryptions and the memory complexity is 2^{12}. Moreover, as the two attacked 6-bit words are independent from each other, one can keep two counters of 6 bits instead of a single counter of 12 bits, which reduces the memory complexity to 2^7.

The implementation of the attack verifies the results given in this section. Using the reference implementation of TWIS and taking $c = 4$, it takes only 15 seconds on a laptop[1] to get the 12 bits of the final subkey. By optimizing the reference code, the attack time can be decreased.

4 Impossible Differential Distinguisher for TWIS

Impossible differential analysis [13,14] is a variant of differential analysis. The fundamental difference between two analysis methods is that in differential cryptanalysis the attacker tries to exploit possible input-output difference pairs to get

[1] 2.2 Ghz Intel Core2Duo Processor, 2 GB Ram, Ubuntu 10.10 64 bit Operating System.

information about the correct key, while in impossible differential cryptanalysis the attacker tries to find events that never occur and use differentials with probability zero, called impossible differentials. In this section, we analyze the security of TWIS with respect to impossible differential cryptanalysis and present a distinguisher of 9.5 rounds.

While building the impossible differential characteristic, we were inspired from the differential characteristic given in Table 2. We combine two differential characteristics with probability one and obtain a contradiction by using the miss-in-the-middle approach [15]. The impossible differential characteristic is depicted in Figure 5, in which "0" denotes the 32-bit word consisting of all zeros.

In the left part of Figure 5, the input difference (0,0,Δy,0), Δy=00800000_x, is proceeded for 4.5 rounds in the forward direction and the difference (Δt,0,0,0), Δt=00200000_x, is obtained. On the other part, starting from the last round of the characteristic, the output difference (Δt,0,0,0) is traced backwards for 5 rounds and (0,0,Δx,0) difference where Δx=01000000_x, is acquired. However, we cannot have (Δt,0,0,0) = (0,0,Δx,0) since both Δt and Δx are non-zero differences. Therefore, (0,0,Δy,0) $\nrightarrow$ (Δt,0,0,0) after 9.5 rounds.

This characteristic can be extended to an impossible differential attack by adding half round to the beginning of the characteristic. By guessing the initial subkeys, wrong values can be eliminated and one will be left with the actual value of the subkeys.

5 Key Related Observations

This section is devoted to the observations on TWIS block cipher. These observations, which are mainly on key scheduling algorithm, include very basic design flaws like actual key size and trivial related key distinguishers that compromise the security of the algorithm.

The most important flaw with the key schedule is that it does not use all bits of the master key. Instead, it uses only 54 bits of the 128-bit key. The first subkey is generated from the first 3 and the last 29 bits of the master key. Each remaining subkey is generated by rotating the modified key 3 bits to the left. So, in order to generate 10 more subkeys, algorithm uses the first 33 bits of the master key. Therefore, key scheduling algorithm uses the first 33 and the last 29 bits of the key to derive 11 subkeys which adds up to 62 bits. Considering the bits eliminated by the S-Boxes in the key scheduling part, the actual key size of the cipher further reduces to 54 bits.

Moreover, the unused 74-bits can be used to distinguish TWIS from other ciphers: if one flips some of the unused key bits, the ciphertext will remain the same. For example, take K and $\overline{K}$ such that $\overline{K_0} = K_0 \oplus 0x10$. After the initial 3-bit rotation, the difference $0x10$ becomes $0x80$. Then, applying S-box to both K_0 and $\overline{K_0}$, one gets $S(K_0 \wedge 0x3f) = S(\overline{K_0} \wedge 0x3f)$ and the difference between the keys will be cancelled after S-box operation. Since there is no other difference between the keys, all the subkeys will be exactly the same. This means, if one encrypts P with K and $\overline{K}$, he gets the same ciphertexts with probability one.

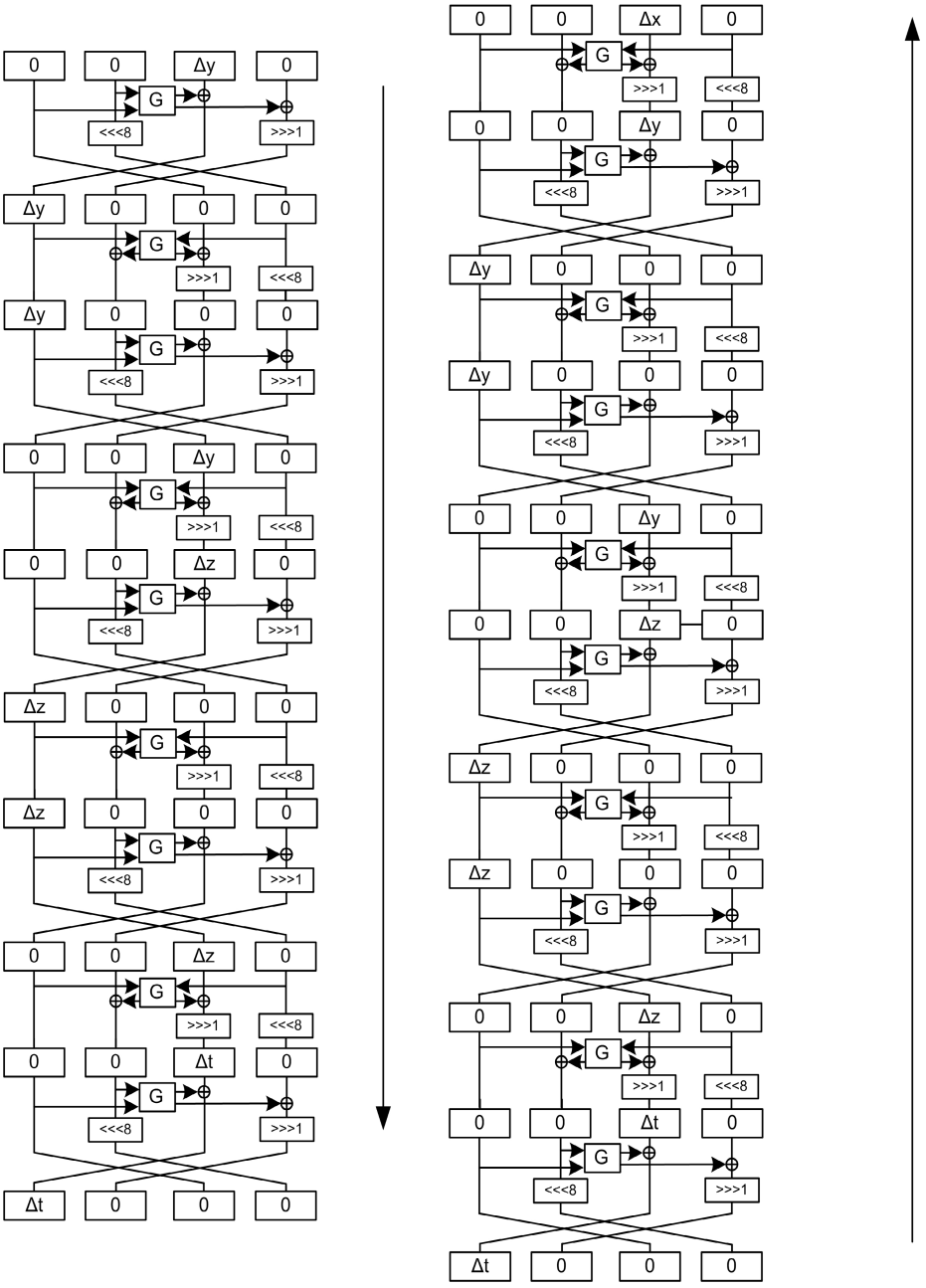


Fig. 5. Impossible Differential Characteristic where Δx=01000000$_x$, Δy=00800000$_x$, Δt=00200000$_x$, and Δz=00400000$_x$

The number of related key distinguishers can be increased by choosing the key differences that coincide the first two bit positions of 8-bit S-box input.

Also, in the data processing part, the data is XORed with the subkey and then S-box is applied to the XORed data. As S-box ignores the first two bits of the 8-bit input, 8 bits of the key is thrown away after this operation. So, the actual subkeys are 24 bits instead of 32 bits.

The key whitening, which is introduced to increase the security, is used in an inappropriate way. Notice that RK_0 is XORed to P_0 as the key whitening which also again XORed to P_0 in the first round inside the G-function. In this way RK_0 will be cancelled in G and it has no effect on the first G-function. So, if one knows the plaintext or specifically P_0, then he also knows the output of the G-function without knowing the key. Therefore, the cipher can be considered as 9.5 rounds.

Furthermore, the choice of final whitening subkeys results in a weakness. If one can determine the whole 32-bits of RK_2 and RK_{10} by attacking the final round, he can also determine the subkeys in between trivially. As the S-box is not invertible (one has to guess the ignored two bits) and there are 3 unknown bits coming from left rotation, it is not possible to go backwards from RK_{10}. Also, one cannot go forwards from RK_2 because of the rotation. However, knowing both, one can determine the missing bits and recover the subkeys $RK_3, RK_4, \ldots, RK_9$ by going backwards from RK_{10}, forwards from RK_2, and checking the known bits. One can recover RK_1, by going backwards from RK_2, with 2^5 computations since there are 3 unknown bits from the rotation and 2 unknown bits from the inverse of the S-box. Similarly, RK_0 can be recovered using RK_1 with 2^5 complexity.

Besides, the diffusion of the key bits into the plaintext is not sufficient. This is a result of using an 8-bit word-wise permutation instead of a bitwise permutation and 8-bit S-box. This enables the attacker to mount an exhaustive search for a 32-bit subkey by dividing it into four 8-bit words without the knowledge of the remaining 24 bits. The complexity of such a search will be $4 \cdot 2^8 = 2^{10}$ instead of 2^{32}. However, in TWIS case, since the S-box ignores two input bits, one can recover the active subkey with $4 \cdot 2^6 = 2^8$ complexity.

6 Conclusion and Future Work

In this paper, we analyze the security of TWIS block cipher against differential, impossible differential attacks. Our results show that 10-round TWIS, when we exclude the final key whitening, is not resistant against differential attack. We recover half of the active key bits with 2^{21} chosen plaintexts. Also, we present distinguishers using the impossible differential technique. This distinguisher can be extended to key recovery attack. Finally, we propose some important observations on the algorithm.

As future a work, we aim to apply the mentioned attacks on full TWIS and mount related-key attacks by using the weaknesses in the key schedule.

References

1. National Institute for Science and Technology (NIST). Advanced Encryption Standard (FIPS PUB 197) (2001), http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf
2. National Institute of Standards and Technology. Federal Information Processing Standard 180-2 Secure Hash Standard (2002), http://csrc.nist.gov/publications/fips/
3. Barker, W.C.: National Institute of Standards, and Technology (U.S.). Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher [electronic resource] U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, Gaithersburg, MD (2004)
4. Rivest, R.L.: The MD5 Message-Digest Algorithm (1992), http://tools.ietf.org/rfc/rfc1321.txt
5. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
6. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
7. Poschmann, A., Leander, G., Schramm, K., Paar, C.: New Light-Weight Crypto Algorithms for RFID. In: ISCAS, pp. 1843–1846 (2007)
8. Hell, M., Johansson, T., Meier, W.: Grain: A Stream Cipher for Constrained Environments. IJWMC 2(1), 86–93 (2007)
9. Ojha, S.K., Kumar, N., Jain, K., Sangeeta: TWIS – A Lightweight Block Cipher. In: Prakash, A., Sen Gupta, I. (eds.) ICISS 2009. LNCS, vol. 5905, pp. 280–291. Springer, Heidelberg (2009)
10. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007)
11. Su, B., Wu, W., Zhang, L., Li, Y.: Full-Round Differential Attack on TWIS Block Cipher. In: Chung, Y., Yung, M. (eds.) WISA 2010. LNCS, vol. 6513, pp. 234–242. Springer, Heidelberg (2011)
12. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)
13. Knudsen, L.: DEAL - A 128-bit Block Cipher. In: NIST AES Proposal (1998)
14. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
15. Biham, E., Biryukov, A., Shamir, A.: Miss in the Middle Attacks on IDEA and Khufu. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 124–138. Springer, Heidelberg (1999)

RSA Vulnerabilities with Small Prime Difference

Marián Kühnel

IT Security Group, RWTH Aachen, Germany
kuehnel@umic.rwth-aachen.de

Abstract. The security of the RSA cryptosystem is based on the assumption that recovering the private key from a public pair is a hard task. However, if the private key is smaller than some bound the system is considered to be insecure. An RSA modulus with a small difference of its prime factors also significantly reduces the overall security. We show that the bound on small private key with respect to small prime difference can be further improved. Therefore, we adapt the technique of unravelled linearization for constructing lattices and although the adapted unravelled linearization is only a method for generating lattices in more elegant way, we yield a benefit compared to known bounds.

Keywords: RSA, unravelled linearization, prime difference, small secret exponent.

1 Introduction

The RSA cryptosystem is currently one of the most widely deployed asymmetric cryptosystems. From a mathematical point of view we generally try to break the RSA cryptosystem either by factorizing the modulus N or by exploiting dependencies in modular equations. In 1990, Wiener [13] demonstrated how one can reveal the private key from public pair if the original private key is smaller than $N^{\frac{1}{4}}$. He also mentioned that his attack may sometimes work for private keys larger than $N^{\frac{1}{4}}$ [13]. This led to an intensive investigation of the modular equation used in private key generation process. Boneh and Durfee improved Wiener's result and presented two approaches based on lattices which can reconstruct the private key smaller than $N^{0.292}$ in polynomial time [1]. This attack was further extended by de Weger in case the modulus is a product of primes with small difference [14]. De Weger derived bounds for both variants where the upper bound for an extended Boneh-Durfee attack was not analyzed in general due to complicated restrictions in forming an adequate lattice. However, Herrmann and May [6] introduced the technique of unravelled linearization[1] which performs a linearization on the modular equation and so exploits induced relations of the linearization itself. These relations were later used for the generation of a lattice. Although their technique did not exceed

[1] The technique of unravelled linearization was originally introduced for exploiting output bits in power generators [5].

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 122–136, 2012.

the bound given by Boneh and Durfee, they provided a new elegant solution to create an appropriate lattice without any complicated restrictions.

In this paper we extend the proposed adapted unravelled linearization method by Herrmann and May [6] for small prime difference in the RSA cryptosystem. Then we discuss properties of the adapted unravelled linearization for small prime difference, compare this technique to other known approaches, explain limitations of geometrically progressive matrices used in the improved Boneh-Durfee attack and finally derivate new improved asymptotic bound where the RSA cryptosystem could still be broken in polynomial time.

The rest of the paper is organized as follows: In Section 2 we will review some preliminaries on lattices followed by the Small Inverse Problem in Section 3 and ideas realized in Boneh-Durfee attack revisited by de Weger in Section 4. Before we present our approach in Section 5 we briefly review the adapted unravelled linearization. In the same Section we also discuss limitations of improved Boneh-Durfee attack and adapted unravelled linearization. Section 6 demonstrates experimental results and the last Section 7 summarizes our work and proposes future work.

2 Short Preliminaries on Lattices

A lattice L is a set of all integer linear combinations of linearly independent vectors $u_1, u_2 \ldots, u_w \in \mathbb{Z}^n$ with $w \leq n$. One can also describe a lattice by its basis matrix B consisting of all the vectors $u_1, u_2 \ldots, u_w$ as row vectors. The dimension of a lattice, denoted $\dim(L)$, is $\dim(L)=w$. The determinant of the lattice L is defined as

$$\det(L) = \prod_{i=1}^{w} ||u_i^*||$$

where $||.||$ denotes the Euclidian norm and $u_1^*, u_2^*, \ldots, u_w^*$ are vectors from applying Gram-Schimdt orthogonalization to the basis vectors. If $w = n$, then the lattice is full rank and the absolute value of the determinant of a lattice basis matrix is equal to the determinant of a lattice. Since all lattices obtained from the adapted unravelled linearization are always full rank and in addition triangular, the determinant is the product of the elements on the diagonal.

Lattices are not restricted to a constant dimension and as the dimension increases, the best reduction algorithm which finds a basis with short vectors of L runs in exponential time. However, for many cryptographic approaches it is sufficient to find only a good approximation of the shortest vector in a basis of a lattice L. One of the most powerful algorithms is the LLL algorithm [8] which reduces and generates a new basis $v_1, v_2 \ldots, v_w$ of the same lattice L with a good approximation of the shortest vector in a reduced basis B' satisfying $||v_1|| \leq 2^{\frac{w}{2}} det(L)^{\frac{1}{w}}$ in polynomial time. The second shortest vector in the reduced basis B' is $||v_2|| \leq 2^w det(L)^{\frac{2}{w-1}}$ [1].

3 The Small Inverse Problem

Recall the RSA and the modulus N which is a product of two primes p and q. Then we denote the prime difference of p and q as $\Delta = |p - q|$. The small prime difference is often rewritten to $\Delta = N^{\beta}$ for $\beta = [\frac{1}{4}, \frac{1}{2}]$ [14]. Note that we start from $\beta = \frac{1}{4}$ because of Fermat's factoring technique [9]. In order to observe dependencies between the small prime difference and small private key d, we define the private key in terms of N, concretely $d = N^{\delta}$ for $\delta \in [0, 1]$.

Lemma 1. *Let p and q be two primes of about the same size of a modulus N and $N^{\beta} = \Delta = |p - q|$. Then $p + q \approx N^{2\beta - \frac{1}{2}}$.*

Proof. We sketch a proof similar to de Weger [14]. We have $\Delta^2 = (p+q)^2 - 4N$ $= (p + q - 2\sqrt{N})(p + q + 2\sqrt{N})$. We know that $2\sqrt{N} < p + q < \frac{3}{\sqrt{2}}\sqrt{N}$. Then

$$p + q - 2\sqrt{N} = \frac{\Delta^2}{p+q+2\sqrt{N}} < \frac{\Delta^2}{4\sqrt{N}}, \;\; p + q + 2\sqrt{N} = \frac{\Delta^2}{p+q-2\sqrt{N}} > \frac{\Delta^2}{4\sqrt{N}}.$$

Therefore, $p + q \approx \frac{\Delta^2}{4\sqrt{N}} \pm 2\sqrt{N} \approx N^{2\beta - \frac{1}{2}}$. □

Boneh and Durfee rewrote the common RSA modular key generation equation into its equivalence and substituted terms $N + 1$ for A and $-(p + q)$ for y.

$$\begin{aligned} ed &= 1 \; (mod \; (p-1)(q-1)) \\ ed &= 1 + x(p-1)(q-1) \\ ed &= 1 + x(N + 1 + (-p-q)) \\ 0 &= 1 + x(A + y) \; (mod \; e) \end{aligned}$$

Now, if modulus N has the same order of magnitude as public key e ($e \approx N^{\alpha}$, $\alpha \approx 1$) then we can solve the Small Inverse Problem for a given polynomial $f(x, y) = 1 + x(A + y) \; (mod \; e)$ satisfying

$$f(x_0, y_0) \equiv 0 \; (mod \; e) \; where \; |x_0| < N^{\delta} \; and |y_0| < N^{2\beta - \frac{1}{2}}.$$

The general idea of solving the Small Inverse Problem is to generate a set of coprime polynomials to the input polynomial $f(x, y)$ which contains the same roots over integers and then use basis reduction and root finding techniques to reveal exact roots over integers. On the other side, if N is significantly larger than e, then also x is and the roots over integers and $(mod \; e)$ are not identical. Then the roots are only linear combinations of e. Fundamental properties for an arbitrary polynomial $f(x, y)$, which has roots over integers, specify the Howgrave-Graham theorem [7] originated from the Coppersmith's method [4].

Theorem 1 (Howgrave-Graham). *Let m be a positive integer and $F(x, y)$ be a bivariate monic polynomial. Suppose that*

(1.) $F(x, y) = 0 \ mod \ N^m$ *and* $|x_0| \leq X$, $|y_0| \leq Y$,
(2.) $||F(xX, yY)|| < \frac{N^m}{\sqrt{dim(L)}}$,

then $F(x_0, y_0)$ has a solution over the integers.

For the proof, we refer the reader to the original paper [7].

4 Geometrically Progressive Matrices in the Boneh-Durfee Attack

The Boneh-Durfee attack is based on the Small Inverse Problem and factorizes a modulus N in the RSA cryptosystem when the private key d is smaller than $N^{0.292}$ [1]. Our aim is to briefly review, examine and join the trivial and improved attack introduced by Boneh and Durfee with de Weger's approach targeting small difference of primes p and q [14]. Hence, let us first recall the initial RSA key generation equation:

$$ed + \underbrace{k}_{x}(\underbrace{N+1}_{A}\underbrace{-(p+q)}_{y}) - 1 = 0.$$

Then the root of the bivariate modular polynomial $f(x, y) = x(A + y) - 1 = 0 \mod e$ is bounded by $|x_0| < X$ and $|y_0| < Y$. Recall that upper bounds X and Y are always smaller than e. Next, if we could generate a set of polynomials satisfying Howgrave-Graham Theorem 1, then we could apply the LLL algorithm on a lattice consisting of polynomials with the same root as $f(x, y)$ and obtain a reduced basis matrix. Row vectors of this reduced basis would lead to a polynomial $F(x_0, y_0) = 0$ with a solution over integers. For the same reason Boneh and Durfee defined the polynomials

$$g_{i,k}(x, y) = x^i f^k(x, y) e^{m-k} \text{ for } k = 0, \ldots, m \text{ and } i = 0, \ldots m - k \text{ and}$$
$$h_{j,k}(x, y) = y^j f^k(x, y) e^{m-k} \text{ for } k = 0, \ldots, m \text{ and } j = 0, \ldots t$$

for some positive integer m and t and within the same roots as $f(x, y)$. Note that the $g_{i,k}(x, y)$ polynomials form the set of all so-called x-shifts while $h_{j,k}(x, y)$ form the set of all y-shifts because they shift each variable x and y, respectively. Then a lattice spanned by the coefficient vectors of these polynomials $g_{i,k}(x, y)$ and $h_{j,k}(x, y)$ where each coefficient vector represented a row of the basis matrix always has a lower triangular structure. The determinant $det(L)$ is determined only by the entries on the diagonal. An example of a lattice L with parameters $m = 2$ and $t = 2$ is depicted in Figure 1. For a better determinant calculation, Boneh and Durfee divided the whole matrix into two submatrices with respect to each shift. Then the determinant corresponding to all x-shifts equals

$$
\begin{array}{c|cccccccccccc}
 & 1 & x & xy & x^2 & x^2y & x^2y^2 & \mathrm{y} & xy^2 & x^2y^3 & y^2 & xy^3 & x^2y^4 \\
e^2 & e^2 & & & & & & & & & & & \\
xe^2 & & e^2X & & & & & & & & & & \\
fe & e & eAX & eXY & & & & & & & & & \\
x^2e^2 & & & & e^2X^2 & & & & & & & & \\
xfe & & eA & & eAX^2 & eX^2Y & & & & & & & \\
f^2 & 1 & 2AX & 2XY & A^2X^2 & 2AX^2Y & X^2Y^2 & & & & & & \\
\hline
ye^2 & & & & & & & \underline{e^2Y} & & & & & \\
yfe & & & eAXY & & & & eY & eXY^2 & & & & \\
yf^2 & & & 2AXY & & A^2X^2Y & 2AX^2Y^2 & Y & 2XY^2 & X^2Y^3 & & & \\
y^2e^2 & & & & & & & & & & \underline{e^2Y^2} & & \\
y^2fe & & & & & & & & eAXY^2 & & eY^2 & \underline{eXY^3} & \\
y^2f^2 & & & & & & A^2X^2Y^2 & & 2AXY^2 & 2AX^2Y^3 & Y^2 & 2XY^3 & X^2Y^4
\end{array}
$$

Fig. 1. Boneh-Durfee basis matrix for m=2, t=2

$$det_x = e^{\frac{1}{3}m(m+1)(m+2)} X^{\frac{1}{3}m(m+1)(m+2)} Y^{\frac{1}{6}m(m+1)(m+2)}.$$

Until now, we have followed the Boneh and Durfee paper [1]. However, if we generalize the determinant for values $|p+q| = N^{2\beta-\frac{1}{2}} \approx e^{2\beta-\frac{1}{2}}$, then the result includes also properties of modulus N with small prime difference [14]. Plugging values N, $e \approx N$, $d = N^{\delta} \approx e^{\delta}$ and $|p+q| = N^{2\beta-\frac{1}{2}} \approx e^{2\beta-\frac{1}{2}}$ into the det_x leads to

$$det_x = e^{\frac{1}{3}m(m+1)(m+2)} e^{\frac{\delta}{3}m(m+1)(m+2)} e^{(\frac{\beta}{3}-\frac{1}{12})m(m+1)(m+2)} = e^{(\frac{\delta}{3}+\frac{\beta}{3}+\frac{1}{4})m^3+o(m^3)}.$$

Boneh, Durfee and later de Weger analyzed and evaluated the bound obtained only from the submatrix related to x-shifts and noticed that one would recover the Wiener's bound $\delta \leq \frac{3}{4} - \beta$. Hence, taking only x-shifts into consideration cannot exceed Wiener's bound. With the help of tedious arithmetic, Boneh with Durfee computed the determinant corresponding to y-shifts and also the whole determinant of a lattice. This led to the bound $\delta < e^{0.284}$ which is equivalent to $\delta < \frac{1}{6}(4\beta+5) - \frac{1}{3}\sqrt{(4\beta+5)(4\beta-1)}$ according to de Weger [14].

Each basis matrix has w row vectors with the largest constant equal to e^m. Therefore, the norm of the first shortest vector v_1 in each reduced basis should not be much larger than $\frac{e^m}{w}$ when we want to solve the Small Inverse Problem. In addition, recall the LLL algorithm which returns a vector $||v_1|| \leq 2^{\frac{w}{2}} det(L)^{\frac{1}{w}}$. Combining these two facts and omitting all negligible values compared to e leads to a weaker condition on determinant. Thus, for the rest of this paper we will assume that each determinant $det(L) \leq e^{mw}$ and each monomial on the diagonale which contributes to the determinant with a factor smaller than e^m improves the overall bound on X and contrarily, each larger value decreases the bound.

Boneh und Durfee examined entries on the diagonal (see Figure 1) and realized that all the underlined values in Figure 1 are exceeding e^m and, therefore, are more destructive than beneficial. Hence, they eliminated row vectors involving these larger values and were able to manage a better bound on private key $\delta < 0.292$ [1]. Unfortunately, by removing some of these rows from the lattice, they violate the advantageous triangular structure. Consequently, the resulting determinant of a new created lattice $L^{'}$ become more complex and much harder to compute. In order to formulate a new determinant $det(L^{'})$ for each non triangular lattice $L^{'}$ they introduced geometrically progressive matrices.

Definition 1. *Let a, b be positive integers and let $C, D, c_0, c_1, c_2, c_3, c_4, \alpha$ be real numbers with $C, D, \alpha \geq 1$. Then an $(a+1)b \times (a+1)b$ matrix M is said to be geometrically progressive with parameters $(C, D, c_0, c_1, c_2, c_3, c_4, \alpha)$ if the following conditions are satisfied for all $i, k = 0, \ldots, a$ and $j, l = 1, \ldots b$:*

1. $|M(i,j,k,l)| \leq C \cdot D^{c_0+c_1 i+c_2 j+c_3 k+c_4 l}$,
2. $M(k,l,k,l) \leq D^{c_0+c_1 i+c_2 j+c_3 k+c_4 l}$,
3. $M(i,j,k,l) = 0$ *for each* $i > k$ *or* $j > k$,
4. $\alpha c_1 + c_3 \geq 0$ *and* $\alpha c_2 + c_4 \geq 0$.

Each geometrically progressive matrix M is equivalent to lattice $L^{'}$ and is divided into two subparts M_x and M_y based on the corresponding shift.

Lemma 2. *The matrix M_y is geometrically progressive with parameters $(m^{2m}, e,$ $m,$ $\delta + 2\beta - \frac{1}{2}, 2\beta - \frac{3}{2}, -1, 1, \alpha)$ for all positive integers m and t [14].*

De Weger verified the geometrically progressive submatrix M_y against all conditions in adapted settings for small prime differences. The first three conditions are always satisfied. However, the last condition holds only for $\delta > 2 - 4\beta$.

$$\underbrace{\alpha(\delta + 2\beta - \frac{1}{2}) - 1 \geq 0 \qquad \alpha(2\beta - \frac{3}{2}) + 1 \geq 0}_{\delta > 2-4\beta}$$

As we mentioned above, with the help of geometrically progressive matrices, Boneh and Durfee managed to improve their bound from $\delta < 0.284$ to $\delta < 0.292$. Also de Weger determined a more general bound on private key $\delta < 1 - \sqrt{2\beta - \frac{1}{2}}$. However, δ must always stay bigger than $2 - 4\beta$. Otherwise, one could not generate a proper geometrically progressive matrix. Since the condition is contradictory for each $\beta < \frac{3}{8}$, de Weger's method exceeds the original bound only for $\delta \leq \frac{1}{2}$ as illustrated in Figure 2. The bold black line represents the final bound achieved by de Weger[2].

[2] Note that for $\beta = \frac{1}{2}$ de Weger would recover Boneh and Durfee results.

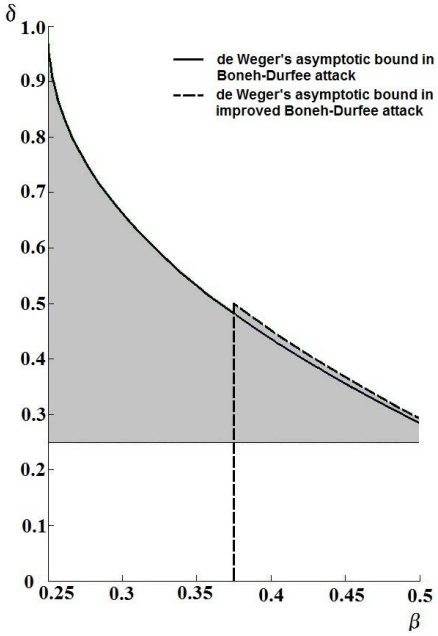


Fig. 2. Both Boneh-Durfee attacks revisited by de Weger

5 The Adapted Unravelled Linearization with Small Prime Difference

Cryptanalysis of RSA with small private key [1] was simplified by Herrmann and May [6]. They revisited an approach called unravelled linearization and manipulate the underlying polynomial so that the geometrically progressive basis matrix described in the previous section becomes triangular with the same asymptotic determinant. Now, we introduce the method of unravelled linearization, explain its always triangular basis matrix, illustrate the similarities to the original Boneh-Durfee attack and discuss an extended attack against the RSA cryptosystem.

Boneh and Durfee improved their attack by removing some of the polynomials that introduced a large factor to the determinant. Consequently, the basis matrix in improved Boneh-Durfee case lost its triangularity and the computing of a determinant became a very complex task. Fortunately, Herrmann and May subtly modified the original equation and performed a more suitable modification of the underlying polynomial which gave them an additional option to eliminate all damaging shifts and led to a low-dimensional sublattice with a triangular basis matrix. The only approach known to the author which effectively reduces row polynomials so that each polynomial in basis matrix contributes to the

determinant with value smaller than e^m and keeps triangular structure at the same time is the unravelled linearization method. In this approach Herrmann and May joined together the monomials of an underlying bivariate polynomial

$$\underbrace{1 + xy}_{u} + Ax \ mod \ e \tag{1}$$

and obtain a linear polynomial $\bar{f}(x, u) = AX + u \ mod \ e$. Then they fixed an integer m and generated x-shifts identical to Boneh-Durfee approach.

$$\bar{g}_{i,k} := x^i \bar{f}^k e^{m-k} \text{ for } k = 0, \ldots, m \text{ and } i = 0, \ldots, m - k$$

Coefficients of $\bar{g}_{i,k}$ represent basis (row) vectors of a lattice. Recall that a lattice spanned only by x-shifts cannot exceed the Wiener bound. Thus, they included y-shifts with an parameter t which will be determined later.

$$\bar{h}_{j,k} := y^j \bar{f}^k e^{m-k} \text{ for } j = 1, \ldots, t \text{ and } k = \left\lfloor \tfrac{m}{t} \right\rfloor j, \ldots, m$$

The crucial observation is that we have a different set of y-shifts polynomials where each polynomial $\bar{h}_{j,k}$ satisfies the second Howgrave-Graham condition in Theorem 1. In addition, each generated polynomial $\bar{h}_{j,k}$ introduces only one new monomial $y^j u^k$ and all other terms are already known from previous polynomials or can be substituted by the term $u - 1$ obtained from the original linearization (1) where Herrmann and May [6] substituted $u = xy + 1$. Then the lattice is formed by a lower triangular matrix and entries on the diagonal indicate the determinant that is now trivial to compute.

Now, we explain the preservation of a triangular basis matrix in the adapted unravelled linearization in detail. First, we consider the x-shifts in the standard lexicographic monomial order. The coefficients of powers of $\bar{f}$ are determined by Pascal's triangle. The left diagonal of Pascal's triangle corresponds to the term $A^n X^n$ for $n = 1, \ldots, k$ while the right diagonal corresponds to the coefficient of U^n, where U denotes the upper bound on u. The next diagonal from the left corresponds to the coefficient of $A^{n-1}X^{n-1}$ and so on. It is also worth noticing that each power of $\bar{f}$ increases the number of terms in a row by one.

$$\begin{array}{l|c}
\bar{f}(xX, uU) & AX \quad U \\
\bar{f}^2(xX, uU) & A^2X^2 \quad 2AXU \quad U^2 \\
\bar{f}^3(xX, uU) & A^3X^3 \quad 3A^2X^2U \quad 3AXU^2 \quad U^3 \\
\bar{f}^4(xX, uU) & A^4X^4 \quad 4A^3X^3U \quad 6A^2X^2U^2 \quad 4AXU^3 \quad U^4 \\
\vdots & \vdots \\
\bar{f}^n(xX, uU) & A^nX^n \quad \binom{n}{1}A^{n-1}X^{n-1}U \quad \ldots \quad \binom{n}{n-1}AXU^{n-1} \quad U^n
\end{array}$$

Fig. 3. Coefficients of power of $\bar{f}^n$

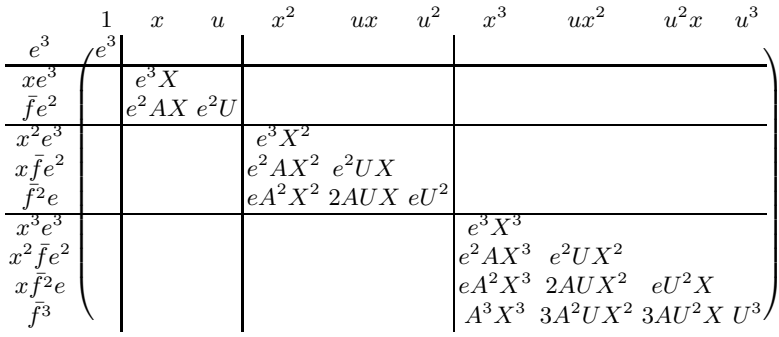


Fig. 4. A lattice only with x-shifts ($m = 3$)

One can rewrite each polynomial $\bar{f}^l$ in Pascal's triangle by a polynomial with smaller power $\bar{f}^{l-l'}$ for $k = 0, \ldots, m$ multiplied by x^i for $i = 0, \ldots, m-k$ and adding additionally several terms containing U. The total degree of a polynomial $\bar{f}^l$ is then $l - l' + i$ and the number of missing terms involving U is equal to i. These coefficients are $U^l, \binom{l}{1}AXU^{l-1}, \ldots, \binom{l}{l-1}A^{i-1}X^{i-1}U$. The x-shifts, which are split into sub-blocks depending on the total degree of $x^i\bar{f}^{l-l'}$, form a collection of polynomials consisting of all combinations of x and $\bar{f}$ so that the sum of exponents is n for n-th sub-block starting from $n = 0$. In addition, although polynomials in each sub-block have a different number of terms, they always form a triangular submatrix. A lower triangular submatrix generated from $\bar{g}_{i,k}$ for parameter $m = 3$ is represented in Figure 4.

The y-shifts are more complex and, therefore, we clarify the dependence of consecutive polynomials by an expansion of $y^j\bar{f}^l e^{m-l}$ for $j = 1, \ldots, t$. First we state that each polynomial $y^{j+1}\bar{f}^l e^{m-l}$ has only one new monomial compared to $y^j\bar{f}^l e^{m-l}$ and then the same for $y^j\bar{f}^{l+1}e^{m-l-1}$. In order to gain a better focus on variables x, y and u, we will omit all constants and upper bounds. It is also worth noticing that we first perform y-shifts for lower powers of y and $\bar{f}$. Thus, our sufficient condition is to check whether there is only one new term larger[3] than the monomials we already know from previous polynomials or not. This restriction is also called an increasing pattern [3]. Let

$$y^j\bar{f}^l = u^l y^j + u^{l-1}xy^j + \ldots + ux^{l-j}y^j + x^l y^j.$$

Then

$$\begin{aligned} y^{j+1}\bar{f}^l &= u^l y^{j+1} + u^{l-1}xy^{j+1} + \ldots + x^l y^{j+1} \\ &= u^l y^{j+1} + u^{l-1}y^j \cdot xy + \ldots + x^{l-j}(xy)^j \\ &= u^l y^{j+1} + u^{l-1}y^j(u-1) + \ldots + x^{l-j}(u-1)^j \\ &= u^l y^{j+1} + u^l y^j - u^{l-1}y^j + \ldots + u^j x^{l-j} - u^{j-1}x^{l-j} \ldots \pm x^{l-j} \end{aligned}$$

[3] Terms are in lexicographic monomial order.

$$
\begin{array}{c}
 \\
e^2 \\
xe^2 \\
\bar{f}e \\
x^2e^2 \\
x\bar{f}e \\
\bar{f}^2 \\
y\bar{f}e \\
y\bar{f}^2 \\
y^2\bar{f}^2
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc}
1 & x & u & x^2 & ux & u^2 & uy & u^2y & u^2y^2 \end{array} \\
\begin{pmatrix}
e^2 & & & & & & & & \\
 & e^2xX & & & & & & & \\
 & eAxX & euU & & & & & & \\
 & & & e^2x^2X^2 & & & & & \\
 & & & eAx^2X^2 & euUxX & & & & \\
 & & & A^2x^2X^2 & 2AuUxX & u^2U^2 & & & \\
-eA & & eAuU & & & & uUyY & & \\
 & -A^2xX & -2AuU & & A^2uUxX & 2Au^2U^2 & & u^2U^2yY & \\
A^2 & & -2A^2uU & & & A^2u^2U^2 & -2AuUyY & 2Au^2U^2yY & u^2U^2y^2Y^2
\end{pmatrix}
\end{array}
$$

Fig. 5. A lattice using unravelled linearization with parameters $m = 2$ and $t = 2$

Indeed, the equation introduces exactly one new monomial $u^l y^{j+1}$ compared to $y^j \bar{f}^l$. The sum of all exponents in all the other terms is smaller than $j + l$. Another way to acquire y-shifts is to increase the power of $\bar{f}$.

$$
\begin{aligned}
y^j \bar{f}^{l+1} &= u^{l+1}y^j + u^l x y^j + \ldots + x^{l+1}y^j \\
&= u^{l+1}y^j + u^l y^{j-1}(u-1) + \ldots + x^{l+1}y^j \\
&= u^{l+1}y^j + u^{l+1}y^{j-1} - u^l y^{j-1} + \ldots + x^{l+1}y^j
\end{aligned}
$$

The polynomial $y^j \bar{f}^{l+1}$ contributes to the y-shifts with $u^{l+1}y^j$, while the monomials $u^{l+1}y^{j-1}$, $-u^l y^{j-1}$, ... emerge first in other polynomials e.g. $y^{j-1}\bar{f}^{l+1}$, $y^{j-1}\bar{f}^l$, etc. The above statements are only true for $l + 1 \geq j$. Otherwise, we would get $x^{l+1}y^j = (xy)^{l+1}y^{j-l-1} = (u-1)^l y^{j-l-1} - \ldots \pm y^{j-l-1}$ in the last monomial. And as we can see in Figure 5, our y-shifts contribute to the basis matrix only with monomials in a form $y^j u^l$ due to j starting from index one and not zero. Another proof, however not as detailed as the aforementioned one, is given in [6].

Notice that if polynomial $y^j \bar{f}^k$ for $k \in \{\lfloor \frac{m}{t} \rfloor j, \ldots, m\}$ is an y-shift, then all of $y^{j-i}\bar{f}^{k-i}$ for $k - i \in \{\lfloor \frac{m}{t} \rfloor j - i, \ldots, m - i\}$ and also $k - i \in \{\lfloor \frac{m}{t} \rfloor (j-i), \ldots, m\}$ are already included in a lattice. A relation between the two different $k - i$ sets states the following inequality [6]

$$
\begin{gathered}
\lfloor \tfrac{m}{t} \rfloor (j-i) \leq \lfloor \tfrac{m}{t} \rfloor j - i \\
i(1 - \lfloor \tfrac{m}{t} \rfloor) \leq 0
\end{gathered}
$$

$$
\left\lfloor \frac{m}{t} \right\rfloor \geq 1. \tag{2}
$$

We proceed with calculating the determinant of the lattice according to Herrmann and May [6]. They also performed rounding compared to large m and t in exponent and suggested a new variable $\tau = \frac{t}{m}$ which avoided long, tedious computations. Then the determinant of the lattice $det(L)$ corresponded to

$$
det(L) < X^{s_x} Y^{s_y} U^{s_u} e^{s_e} \tag{3}
$$

where s_x, s_y, s_u, and s_e are the contributions of the upper bounds X,Y,U and e from all polynomials defined by $g_{i,k}^{-}$ and $h_{j,k}^{-}$.

$$
\begin{aligned}
s_x &= \sum_{k=0}^{m} \sum_{i=0}^{m-k} i = \frac{1}{6}m^3 + o(m^3) \\
s_y &= \sum_{j=1}^{\tau m} \sum_{k=\frac{1}{\tau}j}^{m} j = \frac{\tau^2}{6}m^3 + o(m^3) \\
s_u &= \sum_{k=0}^{m} \sum_{i=0}^{m-k} k + \sum_{j=1}^{\tau m} \sum_{k=\frac{1}{\tau}j}^{m} k = \left(\frac{1}{6} + \frac{\tau}{3}\right) m^3 + o(m^3) \\
s_e &= \sum_{k=0}^{m} \sum_{i=0}^{m-k} (m-k) + \sum_{j=1}^{\tau m} \sum_{k=\frac{1}{\tau}j}^{m} (m-k) = \left(\frac{1}{3} + \frac{\tau}{6}\right) m^3 + o(m^3) \\
dim(L) &= \sum_{k=0}^{m} \sum_{i=0}^{m-k} 1 + \sum_{j=1}^{\tau m} \sum_{k=\frac{1}{\tau}j}^{m} 1 = \left(\frac{1}{2} + \frac{\tau}{2}\right) m^2 + o(m^2)
\end{aligned}
$$

The plus operator in s_u and s_e implies joining together the contributions from both shift sets. Recall that our main goal is to examine the impact of adapted unravelled linearization on small prime differences. Therefore, we adjust the original upper bounds $X = N^{\delta}$, $Y = N^{\frac{1}{2}}$, $U = N^{\delta+\frac{1}{2}}$ from [6] for $Y = N^{2\beta-\frac{1}{2}}$ and $U = N^{\delta+2\beta-\frac{1}{2}}$ and so involve the small prime difference of p and q. We do not need to update $X = N^{\delta}$, since we operate only with prime difference affecting only y-shifts. This leads us to a more general determinant. For more details on determinant calculation we refer the reader to [6].

$$
\begin{gathered}
det(L) < e^{dim(L)m}\ X^{s_x} Y^{s_y} U^{s_u} e^{s_e} \leq e^{dim(L)m} \\
\delta s_x + (2\beta - \tfrac{1}{2})s_y + (\delta + 2\beta - \tfrac{1}{2})s_u + s_e \leq dim(L)m \\
m^3 \left(\tfrac{\delta}{6} + (2\beta - \tfrac{1}{2})\tfrac{\tau^2}{6} + (\delta + 2\beta - \tfrac{1}{2})(\tfrac{1}{6} + \tfrac{\tau}{3}) + \tfrac{1}{3} + \tfrac{\tau}{6} - \tfrac{1}{2} - \tfrac{\tau}{2}\right) \leq 0 \\
m^3(\tau^2(4\beta - 1) + x(4\delta + 8\beta - 6) + 4\delta + 4\beta - 3) \leq 0
\end{gathered}
$$

which is minimal for $\tau = \frac{3-2\delta-4\beta}{4\beta-1}$. Plugging the optimized τ back to the equations leads to

$$
\begin{gathered}
m^3(-4\delta^2 + 8\delta + 8\beta - 6) \leq 0 \\
\delta \leq 1 - \sqrt{2\beta - \tfrac{1}{2}}.
\end{gathered}
$$

We let the reader to check that for $\beta = \frac{1}{2}$ one will recover the bound $\delta \leq 1 - \frac{\sqrt{2}}{2} \approx 0.292$ from the original improved Boneh-Durfee attack.

Although with the aid of unravelled linearization we are able to avoid progressive matrices, for a successful generation of a triangular basis matrix from polynomials $\bar{g}_{i,k}$ and $\bar{h}_{j,k}$, $\tau = \frac{t}{m}$ should remain positive and smaller than one. Otherwise, we would get more than one new term and destroy the triangular structure in the y-shifts. Recall that the adapted unravelled linearization can only be applied for a triangular basis matrix (see Equation 2).

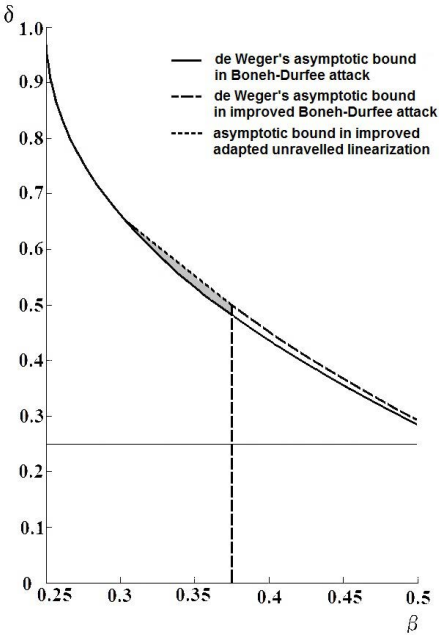


Fig. 6. Improved bound with respect to small prime difference

Then $\frac{3-2\delta-4\beta}{4\beta-1} \leq 1$ implies that our condition on τ is fulfilled only for interval $\beta \in [\frac{3}{8}, \frac{1}{2}]$ or analogously for $\delta \leq \frac{1}{2}$. Recall that these intervals can also be achieved by an improved Boneh-Durfee attack with small prime differences [14] and that geometrically progressive matrices are strictly defined for $\delta \leq 0.5$ [14]. On the other hand, our adapted unravelled linearization solution for small prime difference has also one restriction ($\tau \leq 1$) which is satisfied for values $\delta \leq 0.5$ and after reaching this point $\delta = 0.5$ we are "stopped" only due to the inability of generating an adequate τ. However, if we continue with $\delta > 0.5$ for constant $\tau = 1$ and omit that the method won't be optimized anymore, then we obtain a subtle benefit. The exact impact on boundary function is shown in Figure 6. The shaded area, starting from $\beta < 0.375$, depicts the concrete advantage compared to the (improved) Boneh-Durfee attack and de Weger's result. The dotted line shows the improved boundary function on δ. Since the new boundary function with unoptimized τ is only linear, our approach intersects the lower Boneh-Durfee boundary function at the point $\beta = 0.3125$ and $\delta = 0.625$, which seems to be more suitable for very small prime differences. Hence, from this point on we continue with the lower Boneh-Durfee function.

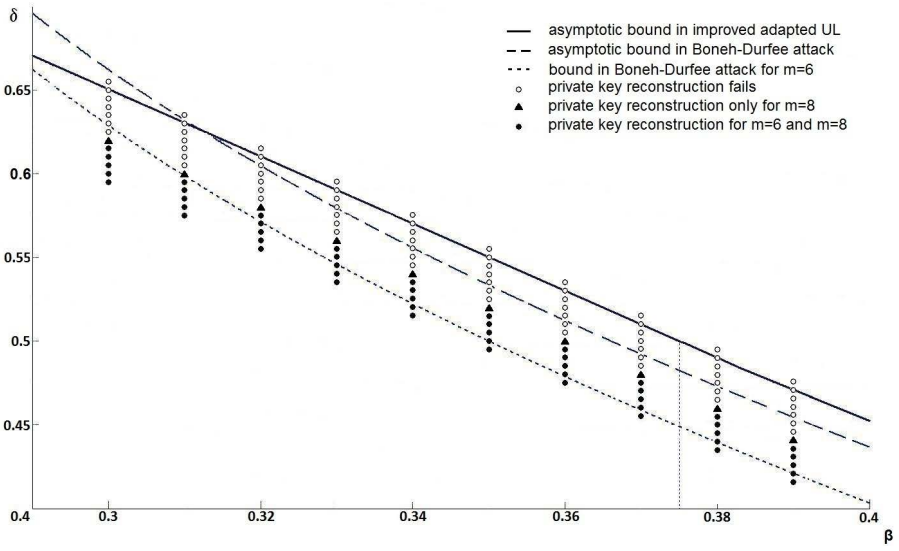


Fig. 7. Improved bound on δ for parameters $m = t = 6$ and $m = t = 8$

6 Experimental Results

Our implementation of unravelled linearization specialized for small prime difference reduces a basis matrix by the LLL algorithm, extracts the first two shortest vectors and computes the original $p + q$ value. Since our approach mainly deals with low level arithmetic functions over big integers we include the NTL library [11] which claims to provide one of the fastest polynomial arithmetic and lattice reductions. In our case, we chose the block Korkin-Zolotarev reduction [12] which yields better quality of the reduced matrix than the classical LLL algorithm. Unfortunately, the NTL library does not support bivariate modular polynomials. Therefore, we are forced to use an additional computer algebra system called PARI/GP [10] designed for fast computations in number theory.

We prove our theoretical assumptions by running dozens of experiments. All tests ran on a 2.66 GHz Intel Xeon processor with 8 GB RAM. For each test, we generate a private key d, primes p and q so that $p + \Delta < q$. We repeat each test several times in order to improve the quality of respective tests.

Recall that the improved Boneh-Durfee attack revisited by de Weger ends at point $\beta = 0.375$ and $\delta = 0.5$ due to the definition of geometrically progressive matrix while the weaker Boneh-Durfee attack continues asymptotically to $\delta = 1$. Hence, our priority is to evaluate the region between where the improved Boneh-Durfee attack revisited by de Weger ends, but adapted unravelled linearzation continues. This area of interest is illustrated in Figure 7. Let us first describe the depicted data. The black bold line represents our asymptotic improved

bound on delta, while the dashed line is the lower asymptotic boundary function given by de Weger. The dotted line denotes its lower version precomputed and optimized for parameter $m = 6$. Circles denotes our tests. We evaluate lattices with parameters $m = t = 6$ and $m = t = 8$ generated by optimized parameters for discussed region. The black-filled circles indicate parameters where we are still able to successfully recover the private key within the smaller lattice with parameter $m = t = 6$. In addition, black-filled triangles determine the benefit which is obtained by choosing a larger lattice with $m = t = 8$. As we can see in Figure 7, results from both lattices are parallel to the derived asymptotic bound for adapted unravelled linearization. Hence, we proved our claim. The empty circles indicate failure solutions in a sense we were not able to obtain the original private key from the reduced basis matrix.

While performing the experiments we observed that the runtime changes every time and is dependent on parameters. Thus, we measured the time during the experiments for respective δ and β. Monotone increasing average times in seconds are shown in the Table 1. Note that β and δ values are identical to circles in Figure 7.

Table 1. Running time for small prime difference

β	0.3	0.31	0.32	0.33	0.34	0.35	0.36
δ	0.62	0.595	0.58	0.56	0.54	0.52	0.5
$m = t = 6$	$96s$	$89s$	$82s$	$73s$	$69s$	$63s$	$56s$
$m = t = 8$	$1418s$	$1292s$	$1175s$	$1102s$	$981s$	$876s$	$817s$

7 Summary

This paper evaluates the region between where the improved Boneh-Durfee ends, but lower Boneh-Durfee attack still continues for small prime differences. Therefore, we reviewed the fundamental ideas behind the original and improved attack and also adapted unravelled linearization. We proved that adapted unravelled linearization holds triangular structure for each $\tau \leq 1$. We described an unique solution which exceeds the de Weger's bound on δ asymptotically although the adapted unravelled linearization lattice is equivalent to the lattice used in the improved Boneh-Durfee attack. Moreover, we did not only demonstrate the correctness of our assumptions, but also give a general overview of running time of the block Korkin-Zolotarev reduction concerning the small primes difference as we could not find any relevant information about it in available literature. Geometrically progressive matrices were not analyzed nor optimized for small prime difference although they could have led to an equivalent bound. This idea is left as an open problem for future research.

Acknowledgments. The author would like to thank Alexander May, Mathias Herrmann from Ruhr University Bochum, Ulrike Meyer, Johannes Barnickel from RWTH Aachen and the anonymous WEWoRC reviewers for their valuable suggestions and comments.

References

1. Boneh, D., Durfee, G.: Cryptanalysis of RSA with Private Key d Less than $N^{0.292}$. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 1–11. Springer, Heidelberg (1999)
2. Blömer, J., May, A.: A Generalized Wiener Attack on RSA. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 1–13. Springer, Heidelberg (2004)
3. Blömer, J., May, A.: Low Secret Exponent RSA Revisited. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 4–19. Springer, Heidelberg (2001)
4. Coppersmith, D.: Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. Journal of Cryptology 10(4), 233–260 (1997)
5. Herrmann, M., May, A.: Attacking Power Generators Using Unravelled Linearization: When Do We Output Too Much? In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 487–504. Springer, Heidelberg (2009)
6. Herrmann, M., May, A.: Maximizing Small Root Bounds by Linearization and Applications to Small Secret Exponent RSA. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 53–69. Springer, Heidelberg (2010)
7. Howgrave-Graham, N.: Finding Small Roots of Univariete Modular Equations Revisited. In: Möhring, R.H. (ed.) WG 1997. LNCS, vol. 1335, pp. 131–142. Springer, Heidelberg (1997)
8. Lenstra, A.K., Hendrik, Lovász, L.: Factoring Polynomials with Rationals Coefficients. Mathematische Annalen 261(4), 515–534 (1982)
9. McKee, J.: Speeding Fermat's Factoring Method. Math. Comput. 68, 1729–1737 (1999)
10. The PARI Group, Bordeaux: PARI//GP, version 2.5.0 (2011), `http://pari.math.u-bordeaux.fr`
11. Shoup, V.: NTL: A Library for Doing Number Theory (2003), `http://www.shoup.net/ntl`
12. Schnorr, C.P.: Block Korkin-Zolotarev Bases and Succesiva Minima (1996)
13. Wiener, M.: Cryptanalysis of Short RSA Secret Exponents. IEEE Transactions on Information Theory 36, 553–558 (1990)
14. De Weger, B.: Cryptanalysis of RSA with Small Prime Difference, Applicable Algebra in Engineering. Communication and Computing 13(1), 17–28 (2002)

Combining Multiplication Methods with Optimized Processing Sequence for Polynomial Multiplier in $GF(2^k)$

Zoya Dyka, Peter Langendoerfer, and Frank Vater

IHP,
Im Technologiepark 25, D-15232 Frankfurt (Oder), Germany
http://www.ihp-microelectronics.com/

Abstract. In this paper we present an approach for optimizing the implementation of hardware multipliers in $GF(2^k)$. We investigate two different strategies namely the reduction of the complexity of the multiplication methods and the combination of different multiplication methods as a means to reduce the area and/or energy consumption of the hardware multiplier. As a means to explore the design space concerning the segmentation of the operands and the selection of the most appropriate multiplication methods we introduce an algorithm which determines the best combination of the multiplication methods. In order to assess the validity of our approach we have benchmarked it against theoretical results reconstructed from literature and against synthesis results using our inhouse 130 nm technology. The former revealed that our designs are up to 32 per cent smaller than those given in literature, the latter showed that our area prediction is extremely accurate.

Keywords: ECC, polynomial multiplication, hardware implementation.

1 Introduction

During recent years elliptic curve cryptography (ECC) has gained significant attention especially for devices such as wireless sensor nodes. Due to their scarce resources hardware implementations are considered important, if not the only way to enable strong cryptographic support on such devices. When it comes to hardware implementation of ECC, the polynomial multiplication in $GF(2^k)$ is the operation which is investigated most since it is one of the most complex field operations and executed very often. There exist many multiplication methods (MMs) for polynomials over $GF(2^k)$ that apply segmentation of both k-bit long multiplicands into n parts (terms): the generalized Karatsuba MM [1] for $n > 1$; Karatsuba MM [2] for 2- and Winograd MM [3] for 3-term operands, that are both the special cases of the generalized Karatsuba MM; Montgomery MM [4] for 5-, 6- and 7- term operands and many other MMs [5]-[8]. All these MMs require less partial multiplications than the classical MM which in principle helps to reduce the chip parameters area and energy consumption. But these

F. Armknecht and S. Lucks (Eds.): WEWoRC 2011, LNCS 7242, pp. 137–150, 2012.

approaches require more XOR-operations than the classical MM so that the reduction of the number of partial multiplications does not in all cases improve the the chip-parameter of the resulting multiplier. This holds especially true for small operands for which the classical MM is the favorite. The reason for this phenomenon is the fact, that the area of an AND gate is smaller than the area of an XOR gate [9]. Thus, a combination of the classical MM for calculating of small partial products with other MM can improve chip-parameters of the resulting multipliers [10]. An additional means to improve the chip-parameters of the multipliers is the reducing the number of additions (XOR-operations). This reduction can be achieved by using pre-defined processing sequences for additions of partial products [11]. If an optimal combination of several multiplication approaches with a reduced number of XOR-operations is found, the area and energy consumption is reduced significantly. In this paper we discuss two approaches to reduce the complexity of polynomial multipliers which can be applied individually, but that result in an optimal result when used in combination. First we describe our algorithm to determine an optimal combination of MM for a given length of operands. In order to assess the complexity of an implementation proposal we need to provide our algorithm with a means to estimate it. For the assessment of the chip parameters we use the number of XOR and AND gates needed for the implementation. We are aware of the fact that gate properties are technology dependent. By initializing our algorithm with the specific area or energy consumption of used gates it can be applied for each technology. Second we explain how the number of XOR operations can be reduced by applying an optimized processing sequence when summing up the partial products. We apply this approach to 6 MMs and we give their gate complexity with the reduced number of XOR operations. Of course the so optimized MM can be used by our algorithm so that the resulting multiplier also benefits from the reduced number of XOR operations. In addition we thoroughly evaluated our approach with two respects. First we benchmarked it against approaches discussed in literature which applied similar MMs or combinations of MMs respectively. Here we used the provided information to reconstruct theoretical data for multipliers relevant for ECC i.e. for operand lengths up to 600 bit. The comparison with those data reveals that our approach leads to an up to 32 per cent reduced area consumption. Second we synthesized the most promising results using our in-house 130 nm technology to see the quality of our theoretical area assessment. This experiment revealed that there is no measurable deviation. The rest of this paper is structured as follows. In the next section we discuss our means to assess the complexity of a certain multiplication method. In section 3 we introduce our approach to reduce the complexity of the multipliers by determining an optimized processing sequence for the summation of partial products. The following section presents our algorithm for selecting the optimal combination of multiplication methods. The evaluation of our approach against literature and synthesis results is discussed in section 5. The paper concludes with a short summary of our major findings.

2 Complexity of Multipliers

First of all we give the definition of multiplications over extended binary fields $GF(2^k)$. Let $A(x) = \sum_{i=0}^{k-1} a_i \cdot x^i$ and $B(x) = \sum_{i=0}^{k-1} b_i \cdot x^i$ be elements of $GF(2^k)$ and $f(x)$ be the irreducible polynomial of degree k generating $GF(2^k)$. The multiplication of the elements over $GF(2^k)$ can be performed in two steps: the calculation of the polynomial products $C(x)$ of degree $(2k-2)$ and its reduction to the degree k. In this paper we concentrate on the optimization of the first step of the multiplication only, i.e. on the polynomial multiplication:

$$C(x) = A(x) \cdot B(x) = \sum_{i=0}^{2k-2} c_i \cdot x^i, with \;\; c_i = \bigoplus_{i=j+l} a_j \cdot b_l, \;\; \forall j, l < k \qquad (1)$$

The symbol $\bigoplus$ in (1) denotes the Boolean XOR operation.

We describe the complexity of polynomial multiplications (1) by the exact numbers of the Boolean XOR and AND operations, denoted as $\#XOR$ and $\#AND$, respectively. This corresponds to the number of XOR and AND gates of the resulting multipliers. The exact gate complexity (GC) of a certain multiplication method (MM) for k-bit long operands can be expressed by a tuple as follows:

$$GC_k^{MM} = (\#AND; \quad \#XOR) \qquad (2)$$

The optimization parameters of a hardware multiplier, such as its area and the average energy consumption one of a single clock cycle, can be calculated based on its gate complexity and on the area and/or the energy consumption of the used gates ($Area_{AND}, Area_{XOR}$ and E_{AND}, E_{XOR}):

$$\begin{aligned} Area &= \#AND \cdot Area_{AND} + \#XOR \cdot Area_{XOR} \\ Energy &= \#AND \cdot E_{AND} + \#XOR \cdot E_{XOR} \end{aligned} \qquad (3)$$

The reduction of the gate complexity improves directly the area and energy, i.e. the both optimization parameters at the same time. MMs for large k-bit polynomials normally use segmentation of the polynomials into n smaller m-bit terms which are then multiplied. To get the result the partial products are added (i.e. XORed). If this principle of divide and conquer is applied to an k-bit multiplier the resulting ASIC (Application-Specific Integrated Circuit) consist of a certain number of m-bit partial multipliers with their own gate complexity GC_m^{MM}:

$$GC_{k=nm}^{MM} = (\#MULT \cdot GC_m^{MM}; \quad \#XOR) \qquad (4)$$

The number of partial multiplications $\#MULT$ and the number of XOR gates $\#XOR$ depend on the selected multiplication method MM and on the segmentation of the operands, i.e. on the number n. The knowledge of the gate complexity of each partial multiplier as tuple (2) allows to calculate the gate complexity of full k-bit multipliers also as tuple (2). Each partial multiplication can be implemented by any multiplication method or even by any combination of multiplication methods. In order to optimize the complexity of a polynomial

multiplier it is necessary to determine the optimal combination of different MMs. Formula (4) shows the assessment function that allows comparing different MMs given the fact that the segmentation and the type of the used partial multipliers are the same. In the next section we determine the gate complexity according to (4) for 6 MMs using the usual straight forward implementation for adding the partial products and discuss the reduction of the gate complexity using our predefined processing sequence. The algorithm for determining of optimal combination of MMs is presented in section 4.

3 Optimizing Processing Sequences for Polynomial Multiplication Based on the Table Representation

In this section we present how we optimized the processing sequence for different MMs. We determined the optimized processing sequence and its gate complexity according to (4) for the following MMs: classical MM, Karatsuba MM by segmentation of operands into 4 terms [12], Montgomery multiplication formulae for 5-, 6- and 7-term operands [5] and the generalized Karatsuba (genKar) algorithm [1]. In order to do so we represented each MM as a table. This section explains this table representation of a multiplication formula. This table representation (TR) helps to find the same sums of partial products and to prevent their multiple calculation. We explain the table representation of a multiplication formula using the classical MM for 2-bit and for 2-terms long polynomials as example. Table 1 is the table representation of the classical multiplication formula for 2-bit long polynomials (5). The leftmost column of the TR shows all 1-bit long partial products from (5). All other columns correspond to an individual 1-bit long product term c_i, where $0 \leq i \leq 2$. If the product term c_i is calculated by adding a partial product PP_j, the cell of the TR with 'coordinates' (PP_j, c_i) is filled with a $\oplus$. Otherwise the cell is empty.

$$\begin{aligned} C(x) = A(x) \cdot B(x) = a_1a_0 \cdot b_1b_0 = (a_1 \cdot 2 \oplus a_0) \cdot (b_1 \cdot 2 \oplus b_0) = \\ = \underbrace{a_1 \cdot b_1}_{c_2} \cdot 2^2 \oplus \underbrace{(a_0 \cdot b_1 \oplus a_1 \cdot b_0)}_{c_1} \cdot 2^1 \oplus \underbrace{a_0 \cdot b_0}_{c_0} = c_2c_1c_0. \end{aligned} \tag{5}$$

Table 1. TR of the classical MM for 2-bit long polynomials

$a_0 \cdot b_0$			$\oplus$
$a_0 \cdot b_1$		$\oplus$	
$a_1 \cdot b_0$		$\oplus$	
$a_1 \cdot b_1$	$\oplus$		
	c_2	c_1	c_0

TR of the classical multiplication formula for 2-term polynomials looks a bit more complicated, because each operand term A_i or B_i is now m-bit long and

each partial product $A_i \cdot B_j$ is $(2m-1)$-bit long:

$$C(x) = A(x) \cdot B(x) = A_1A_0 \cdot B_1B_0 = (A_1 \cdot 2^m \oplus A_0) \cdot (B_1 \cdot 2^m \oplus B_0) = \\ = \underbrace{A_1 \cdot B_1}_{\mathrm{CS}_2} \cdot 2^{2m} \oplus \underbrace{(A_0 \cdot B_1 \oplus A_1 \cdot B_0)}_{\mathrm{CS}_1} \cdot 2^m \oplus \underbrace{A_0 \cdot B_0}_{\mathrm{CS}_0} = C_3C_2C_1C_0. \quad (6)$$

The highest segment C_3 of the polynomial product $C(x)$ in (6) is $(m-1)$-bit long and all other segments C_i, for $0 \le i \le 2$, are m-bit long. All these product segments can be calculated from segments of $(2m-1)$-bit long terms CS_i. Formula (7) represents each term CS_i as the sum of its segments: the segment consisting of the most significant bits $CS_i[1]$ is $(m-1)$-bit long and the other segment $CS_i[0]$ is m-bit long:

$$CS_i = CS_i[1] \cdot 2^m \oplus CS_i[0] \quad (7)$$

Each partial product $A_i \cdot B_j$ in (6) can be represented in the same way:

$$A_i \cdot B_j = A_iB_j[1] \cdot 2^m \oplus A_iB_j[0] \quad (8)$$

Formula (9) represents the polynomial product from (6) as the sum of the partial product segments in the notation of (8):

$$C(x) = \underbrace{(A_1B_1[1] \cdot 2^m \oplus A_1B_1[0])}_{\mathrm{CS}_2} \cdot 2^{2m} \oplus \\ \oplus \underbrace{\Big((A_0B_1[1] \cdot 2^m \oplus A_0B_1[0]) \oplus (A_1B_0[1] \cdot 2^m \oplus A_1B_0[0])\Big)}_{\mathrm{CS}_1} \cdot 2^m \oplus \\ \oplus \underbrace{(A_0B_0[1] \cdot 2^m \oplus A_0B_0[0])}_{\mathrm{CS}_0} = \\ = \underbrace{A_1B_1[1]}_{\mathrm{C}_3} \cdot 2^{3m} \oplus \underbrace{(A_1B_1[0] \oplus A_0B_1[1] \oplus A_1B_0[1])}_{\mathrm{C}_2} \cdot 2^{2m} \oplus \\ \oplus \underbrace{(A_0B_0[1] \oplus A_0B_1[0] \oplus A_1B_0[0])}_{\mathrm{C}_1} \cdot 2^m \oplus \underbrace{A_0B_0[0]}_{\mathrm{C}_0}. \quad (9)$$

Table 2 is the table representation of formula (9). The leftmost column shows both segments $PP_j[0]$ and $PP_j[1]$ of all partial products PP_j, i.e. $PP_j[d]$, where $1 \le j \le \#MULT$, $d \in \{0,1\}$. All other columns correspond to an individual product term C_i, where $0 \le i \le 2n-1$. If the product term C_i is calculated by adding a partial product segment $PP_j[d]$, the cell of the TR with 'coordinates' $(PP_j[d], C_i)$ is filled with a $\oplus$. Otherwise the cell is empty.

The table representation of a multiplication formula is much easier to assess than its algebraic representation if the length k (or the segmentation n)

Table 2. TR of the classical MM, 2-term operands

A_0B_0 [0]				$\oplus$
A_0B_0 [1]			$\oplus$	
A_0B_1 [0]			$\oplus$	
A_0B_1 [1]		$\oplus$		
A_1B_0 [0]			$\oplus$	
A_1B_0 [1]		$\oplus$		
A_1B_1 [0]		$\oplus$		
A_1B_1 [1]	$\oplus$			
	C_3	C_2	C_1	C_0

of operands is big. The reduction of the complexity of a multiplication formula using an optimized calculation sequence will be shown using the TR of the generalized Karatsuba MM for 4-bit long operands (see (10)) as example.

$$
\begin{aligned}
C(x) = A(x) \cdot B(x) &= a_3a_2a_1a_0 \cdot b_3b_2b_1b_0 = c_6c_5c_4c_3c_2c_1c_0 = \\
&= \underbrace{a_0 \cdot b_0}_{c_0} \oplus \underbrace{(a_0 \cdot b_0 \oplus a_1 \cdot b_1 \oplus (a_0 \oplus a_1) \cdot (b_0 \oplus b_1))}_{c_1} \cdot 2^1 \oplus \\
&\quad \oplus \underbrace{(a_0 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_2 \cdot b_2 \oplus (a_0 \oplus a_2) \cdot (b_0 \oplus b_2))}_{c_2} \cdot 2^2 \oplus \\
&\quad \oplus \underbrace{\begin{pmatrix} a_0 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_3 \cdot b_3 \oplus \\ \oplus (a_0 \oplus a_3) \cdot (b_0 \oplus b_3) \oplus (a_1 \oplus a_2) \cdot (b_1 \oplus b_2) \end{pmatrix}}_{c_3} \cdot 2^3 \oplus \\
&\quad \oplus \underbrace{(a_1 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_3 \cdot b_3 \oplus (a_1 \oplus a_3) \cdot (b_1 \oplus b_3))}_{c_4} \cdot 2^4 \oplus \\
&\quad \oplus \underbrace{(a_2 \cdot b_2 \oplus a_3 \cdot b_3 \oplus (a_2 \oplus a_3) \cdot (b_2 \oplus b_3))}_{c_5} \cdot 2^5 \oplus \underbrace{a_3 \cdot b_3}_{c_6} \cdot 2^6
\end{aligned}
\tag{10}
$$

Table 3 is the TR of formula (10). In the rest of this section we denote the first 4 lines of Table 3 as part1 of the TR and remaining lines as part 2 of the TR. The usual way to calculate product segments is the addition of all their partial products. This requires 15 XOR gates in total for the calculation of all c_i, for $0 \leq i \leq 6$ (see Table 3). In this case each product segment is calculated individually, i.e. no already calculated partial sums are re-used. In the rest of this article we call this way to calculate product segments *separately*. Figure 1 illustrates this calculation way.

Another way to calculate these product segments is the *iterative* calculation. It exploits the fact, that many TR-columns contain the same sums of partial products. For example, each column value of the part 1 of the TR (see Table 3) can be calculated as a sum of its neighbor-columns and only one additional partial product as it is shown in Figure 2.

Table 3. TR of the generalized Karatsuba multiplication formula for 4-bit long operands

$a_0 \cdot b_0$				⊕	⊕	⊕	⊕
$a_1 \cdot b_1$			⊕	⊕	⊕	⊕	
$a_2 \cdot b_2$		⊕	⊕	⊕	⊕		
$a_3 \cdot b_3$	⊕	⊕	⊕	⊕			
$(a_0 \oplus a_1) \cdot (b_0 \oplus b_1)$						⊕	
$(a_0 \oplus a_2) \cdot (b_0 \oplus b_2)$					⊕		
$(a_0 \oplus a_3) \cdot (b_0 \oplus b_3)$				⊕			
$(a_1 \oplus a_2) \cdot (b_1 \oplus b_2)$				⊕			
$(a_1 \oplus a_3) \cdot (b_1 \oplus b_3)$			⊕				
$(a_2 \oplus a_3) \cdot (b_2 \oplus b_3)$		⊕					
	c_6	c_5	c_4	c_3	c_2	c_1	c_0

Part 1

Part 2

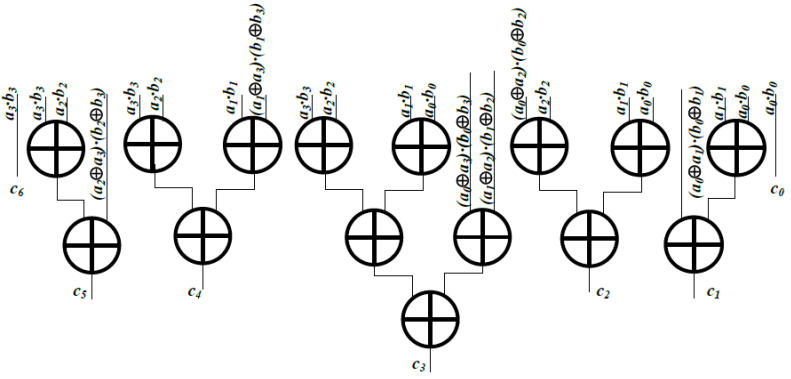


Fig. 1. Usual calculation way: each product segment c_i, for $0 \leq i \leq 6$, in Table 3 is calculated *separately* of other product segments. It requires 15 XOR in total

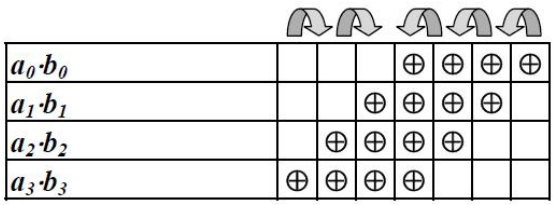


Fig. 2. Optimized processing sequence for *iterative* summation of partial products from part 1 of Table 3. The arrows indicate the sequence in which columns are used to calculate their neighbors. The summation requires only 5 XOR gates: $s_1 = a_0 \cdot b_0 \oplus a_1 \cdot b_1, s_2 = s_1 \oplus a_2 \cdot b_2, s_3 = s_2 \oplus a_3 \cdot b_3, s_4 = a_3 \cdot b_3 \oplus a_2 \cdot b_2, s_5 = s_4 \oplus a_1 \cdot b_1$

The optimized processing sequence for calculating the product segments of the whole TR (see Table 3) can be described by the following two steps:

1. *iterative* calculation of all column values from part 1
2. addition of partial products from part 2 to the values obtained in step 1

The summation using the processing sequence shown in Figure 2 requires only 5 XOR-gates. In addition 6 XOR-gates are needed for the summation of partial products from part 2 of the TR. So, it requires only 11 XOR-gates in total. Figure 3 illustrates the calculation of the product segments of the whole TR using the optimized processing sequence. If segments of operands are more than 1 bit long, the table representation looks a bit more complicated, but the optimized processing sequence which prevents the multiple calculation of the same sums of partial products can be determined. Only for the classical MM the processing sequence cannot be optimized. We determined the optimized processing sequence for all MMs listed in Table 4. Table 4 shows the gate complexity of these MMs, with and without using the pre-defined optimized processing sequences. The first column of Table 4 shows for each MM the used segmentation, i.e. shows in how much terms both multiplicands are segmented. Note, that the 2-term Karatsuba MM and the 3-term Winograd MM are special cases of n-term generalized Karatsuba MM. For the Karatsuba MM with segmentation of operands into 4 terms we use the optimized processing sequence presented by

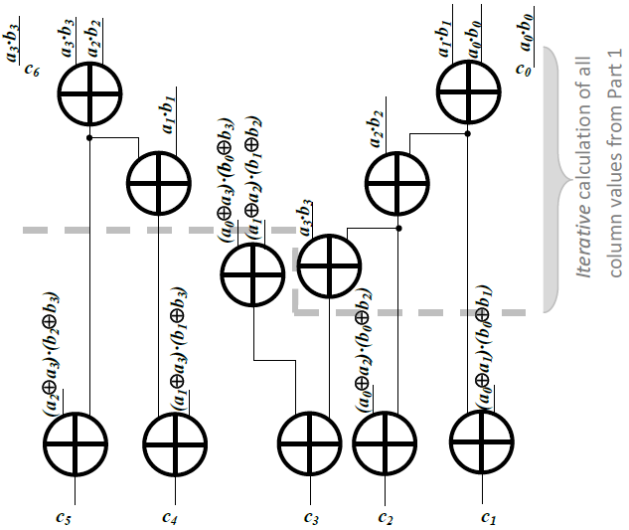


Fig. 3. Calculation of product segments of TR (see Table 3) using the optimized predefined processing sequence. This calculation way exploits the fact, that many TR-columns contain the same sums of partial products. It requires only 11 XOR-gates together.

us in [12]. We use the gate complexity of all these MM with optimized processing sequence in Algorithm 1 to find the optimal combination of MMs. The next section describes this algorithm.

Table 4. Gate Complexity of investigated MMs

n	**MM**	$\#MULT$	$\#XOR$, **usual calculation way**	$\#XOR$, **calculation with optimized processing sequence**
2	Karatsuba	3	$8m-4$	$7m-3$
3	Winograd	6	$24m-11$	$18m-6$
4	Karatsuba	9	$40m-16$	$34m-11$
5	Montgomery	13	$94m-40$	$66m-23$
6	Montgomery	17	$130m-57$	$96m-34$
7	Montgomery	22	$184m-80$	$133m-47$
n	classical	n^2	$2mn(n-1)-n^2+1$	$2mn(n-1)-n^2+1$
n	gen.Kar.	$\frac{n^2+n}{2}$	$4mn(n-1)-\frac{3n^2-n}{2}+1$	$m(2n^2+n-3)-\frac{n^2+n}{2}$

4 Algorithm for Determining Optimal Combination of MMs

Designing an optimal k-bit multiplier requires to know the gate complexity of those m-bit partial multipliers, which might be used. These m-bit partial multipliers should already be optimized. The same holds true for optimizing the m-bit partial multiplier and so on. So, to determine the optimal combination of MMs Algorithm 1 is starting from 1-bit polynomials to up to k-bit polynomials. It is essential to determine all possible segmentations for each length of polynomials i, $1 < i \leq k$. The gate complexity of i-bit multipliers is calculated for each investigated MM considering all possible segmentations of i. The final loop of Algorithm 1 tests, whether the area of the recently determined optimal i-bit multiplier is smaller than the area of the $(i-1)$-bit multiplier, and if so, it replaces the latter with the former. This condition is checked for all s-bit multipliers of smaller length, which are used as optimal m-bit multipliers in the following processing steps of Algorithms 1. This idea improves the results of the combining algorithm from [10] where the optimal combination of MMs for FPGA-implementation of polynomial multiplication is searched. Note that our Algorithm 1 is designed for searching the optimal combination of MMs for ASICs. In addition it can be used for the search of optimal combination of MMs for FPGA after initializing the variables: $Area_{AND} = 1, Area_{XOR} = 1$. In this case the calculated area of multipliers represents the total number of AND and XOR gates.

While Algorithm 1 itself is technology independent, there are two technology dependent parameters to be considered. Technology dependent values such as $Area_{AND}, Area_{XOR}$ (or E_{AND}, E_{XOR} respectively) are input variables. Depending on the optimization goal - area or energy - we use respective parts of eq. (3).

Algorithm 1

Input : $Area_{AND}, Area_{XOR}$ //if optimization goal is area
$MM = \{MM_{clas}, MM_2, MM_3, ...\}$ //set of MMs with optimized //processing sequence
Output : $MM_{opt}(i),\ 1 \leq i \leq k$ //i.e. area, GC and the combination of MMs //that results this area
Initialization : $MM_{opt}(1) = MM_{clas}(1);\ \ MM_{opt}(i) = empty,\ 2 \leq i \leq k$
Calculation :
for $2 \leq i \leq k$ //all operands of smaller length
for $n | 2 \leq n \leq i, n\ divides\ i$ //all possible segmentations
for *each element* MM_j *from* MM
calculate $Area(GC_{i=nm}^{MM_j})$ //see (2), (3), (4) and Table 4
if $MM_{opt}(i) = empty\ or\ Area(GC_{i=nm}^{MM_j}) < Area(GC_i^{MM_{opt}(i)})$
$MM_{opt}(i) = MM_j$
end if
end for
end for
for $s | i > s > 0$ //all operands of smaller length
if $Area(GC_s^{MM_{opt}(s)}) > Area(GC_{s+1}^{MM_{opt}(s+1)})$
$MM_{opt}(s) = MM_{opt}(s+1)$
end if
end for
end for

5 Evaluation of the Optimization Results

In order to benchmark our results we are using results from [1][1] and [10]. Since we are mainly interested in ECC we are investigating polynomials with a length of up to 600 bits. In order to compare our results with those presented in [1] and

[1] By results of [1] we denote those provided in [1] for the recursively applied generalized Karatsuba MM with minimal number of AND and XOR gates.

[10] we reconstructed their results for polynomials with a length up to 600 bits. In order to do so we strictly followed the calculation procedures described in [1] and [10]. The reconstructed data for polynomials up to 128 bits are the same as those given in [1] and [10], which makes us confident that our reconstruction yields correct results.

Figure 4 shows the calculated area of multipliers for polynomial length up to 600 bit for all three approaches. The dashed curve in Figure 4 depicts the chip-area of multipliers that we calculate from [1]. The gray curve in Figure 4 depicts the results from [10]. The area consumption of our approach (black curve, Figure 4) is 32% smaller than the results represented by the dashed curve and about 10% smaller than the results represented by the gray curve. Table 5 shows

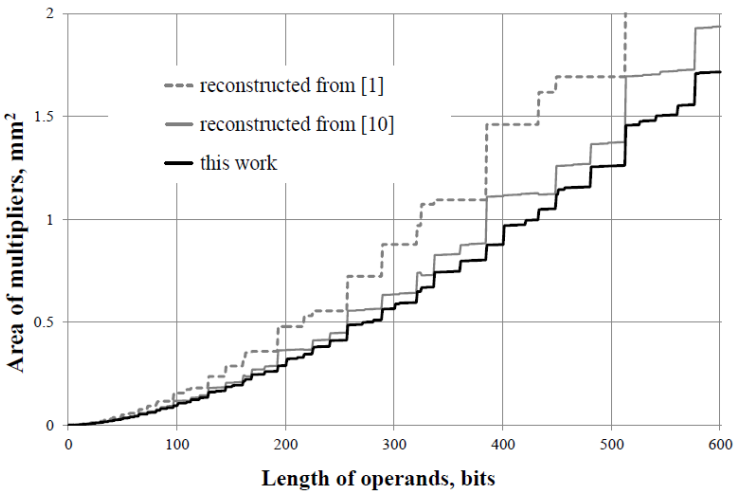


Fig. 4. Calculated area of multipliers for the polynomial length up to 600 bits

the gate complexity and calculated area of multipliers for ECC-relevant operand lengths for all three approaches. Please note that the number of AND and XOR gates of our combinations of MMs are selected based on the results of Algorithm 1, i.e. they reflect the number of gates for the smallest polynomial multipliers for the IHP technology [13]. For this technology the area of AND and XOR gates are 8.069 μm^2 and 13.45 μm^2, respectively. When comparing the results it became apparent that the number of AND gates is smallest for [1]. But our approach and the approach from [10] require by far less XOR gates. This is the reason for the much smaller area of our multipliers. For the evaluation of our theoretical results we synthesized the polynomial multipliers for 3 ECC-relevant lengths of operands: 163, 233 and 283 bits. Table 6 shows the optimal MM-combinations determined by our algorithm and their gate complexities. The area of the multipliers calculated based on their gate complexity and the synthesis results are also given in Table 6. The area of all synthesized multipliers is smaller than the

Table 5. Gate complexity and calculated area of polynomial multipliers for ECC-relevant operand lengths

k, bits	**reconstructed from [1]**			**reconstructed from [10]**			**our combination of MMs with optimized processing sequence**		
	$\#AND$	$\#XOR$	area, mm^2	$\#AND$	$\#XOR$	area, mm^2	$\#AND$	$\#XOR$	area, mm^2
163	4536	23417	0.3516	7938	12820	0.2365	7938	11751	0.2221
233	6561	37320	0.5549	12150	23468	0.4137	12150	21066	0.3814
283	8748	48485	0.7227	13122	34108	0.5646	13122	30091	0.5106
409	17496	98039	1.4598	26244	67420	1.1186	29700	54418	0.9716
571	26244	147755	2.1991	39366	104704	1.7259	37179	93383	1.5560

Table 6. Comparison of theoretical and practical results; the name in brackets after segmentation shows the used MM: 'Kar' means the Karatsuba MM, 'Win' means the Winograd MM and 'clas' means the classical MM

k, bits	**Optimal combination of MMs**	$\#AND$	$\#XOR$	**area,** mm^2	
				theor.	synthes.
163	$4(Kar) \cdot 2(Kar) \cdot 3(Win) \cdot 7(clas)$	7938	11751	0.22	0.20
233	$4(Kar) \cdot 4(Kar) \cdot 3(Win) \cdot 5(clas)$	12150	21066	0.38	0.36
283	$2(Kar) \cdot 4(Kar) \cdot 4(Kar) \cdot 3(Win) \cdot 3(clas)$	13122	30091	0.51	0.48

theoretically calculated area. This can be explain by the following fact: the calculation of the area of multipliers is based on the area of only 2-inputs-AND and 2-inputs-XOR gates. In reality each technology provides also other gate types. For example, the IHP 130 nm technology [13] has also a 3-inputs-XOR gate. The Synopsis-tools [14], that we use for the synthesis of multipliers, apply often one 3-inputs-XOR gate instead two 2-inputs-XOR gates resulting in a reduced area. With the goal to confirm this hypotheses, we implemented a single 8-bit multipliers twice. Both designs of the multiplier are implemented as a combination of the classical MM and the 2-term Karatsuba MM with an optimized processing sequence. First the design was synthesized using only 2-inputs AND and XOR gates. Second the design was synthesized without any restriction of the gate types. Table 7 shows the theoretically calculated area of this multiplier and area of both synthesized versions. The area of the design synthesized using only 2-inputs AND and XOR gates is the same as the theoretically predicted area. The reduction of the area using all available gate types is about 8 per cent. We assume that the percentage of the area reduction, introduced by the tool optimizations, is independent of the size of of the operands. For the evaluation of our theoretical data with the practical results we calculate the relation between areas of all synthesized multipliers. We use the area of the 163-bit multiplier as the "norm". We expect that the relation between theoretical calculated areas of

Table 7. Comparison of theoretical and practical results for 8-bit polynomial multiplier

k, bit	combination of MMs	theoretical data			Area of synthesized designs	
		$\#AND$	$\#XOR$	Area, μm^2	2-inputs AND and XOR gates only	all available gate typs
8	$2(Kar) \cdot 4(clas)$	48	55	1087	$1087\mu m^2$	$995\mu m^2$

Table 8. Area relations for theoretical and synthesized results for multipliers from Table 6

k, bit	area relations	
	theoretical	synthesized
163	1	1
233	1.7	1.7
283	2.3	2.4

multipliers and between areas of synthesized designs are approximately same. Table 8 shows these relations. So the synthesis results confirm the theoretical data.

6 Conclusion

In this paper we have elaborated how an optimal hardware multiplier can be constructed. In order to allow a fair comparison of different multiplication methods we defined the gate complexity, a tuple that represents the number of required AND and XOR gates as the assessment function of the area of the resulting hardware multiplier. In addition we investigated how individual multiplication methods can be optimized by taking into account that some partial products are used to calculate several partial sums. I.e. we determined processing sequences which help to reduce the number of required XOR gate. In order to ensure an optimal combination of MMs we designed and presented an algorithm that determines the optimal segmentation of operands and an optimal combination of multiplication methods from a set of investigated multiplication methods. Please note that the MMs with optimized processing sequences (i.e. with reduced number of XOR gates) are used as input for this algorithm. In order to evaluate our findings we compared our results with results given in literature revealing that our approach leads to a 32 per cent and 10 per cent reduced area consumption compared to [1] and [10] respectively. In addition we compared our area prediction with synthesis results for our inhouse technology. This revealed that there is no measurable deviation if the type of gates used for synthesis is restricted to 2-input XOR and 2 input AND gates. If all types of gate are allowed for synthesis the resulting area is even smaller than the calculated one. To summarize our approach provides extremely accurate prediction of the area consumption of investigated multiplier designs and improves the required area of the multipliers significantly.

References

1. Weimerskirch, A., Paar, C.: Generalizations of the Karatsuba Algorithm for Efficient Implementations. Report 2006/224, Cryptology ePrint Archive (2006), http://eprint.iacr.org/2006/224.pdf
2. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers by automata. Soviet Physics-Doklady 7, 595–596 (1963)
3. Winograd, S.: Arithmetic Complexity of Computations. SIAM (1980)
4. Montgomery, P.L.: Five, Six, and Seven-Term Karatsuba-Like Formulae. IEEE Transactions on Computers 54(3), 362–369 (2005)
5. Fan, H., Hasan, A.: Comments on "Five, Six, and Seven-Term Karatsuba-Like Formulae". IEEE Transactions on Computers 56(5), 716–717 (2007)
6. Sunar, B.: A Generalized Method for Constructing Subquadratic Complexity $GF(2^k)$ Multipliers. IEEE Transactions on Computers 53(9), 1097–1105 (2004)
7. Cenk, M., Koc, C.K., Ozbudak, F.: Polynomial multiplication over finite fields using field extensions and interpolation. In: 19th IEEE Symposium on Computer Arithmetic, pp. 84–91. IEEE Computer Society Press, Portland (2009)
8. Oseledets, I.: Improved n-term Karatsuba-like Formulae in GF(2). IEEE Transactions on Computers (2010), http://doi.ieeecomputersociety.org/10.1109/TC.2010.233
9. Horowitz, P.I., Hill, W.: The Art of electronics. Cambridge University Press, New York (1989)
10. von zur Gathen, J., Shokrollahi, J.: Efficient FPGA-Based Karatsuba Multipliers for Polynomials over $\mathbb{F}_2$. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 359–369. Springer, Heidelberg (2006)
11. Dyka, Z., Langendoerfer, P.: Area efficient hardware implementation of elliptic curve cryptography by iteratively applying Karatsuba method. In: Proc. of the Design, Automation and Test in Europe Conference and Exhibition, vol. 3, pp. 70–75 (2005)
12. Peter, S., Langendoerfer, P.: An Efficient Polynomial Multiplier $GF(2^m)$ and its Application to ECC Designs. In: Proc. of the Design, Automation and Test in Europe Conference and Exhibition, pp. 1253–1258 (2007)
13. Innovations for High Performance Microelectronics, http://www.ihp-microelectronics.com/
14. Synopsis, http://www.synopsys.com/

Author Index